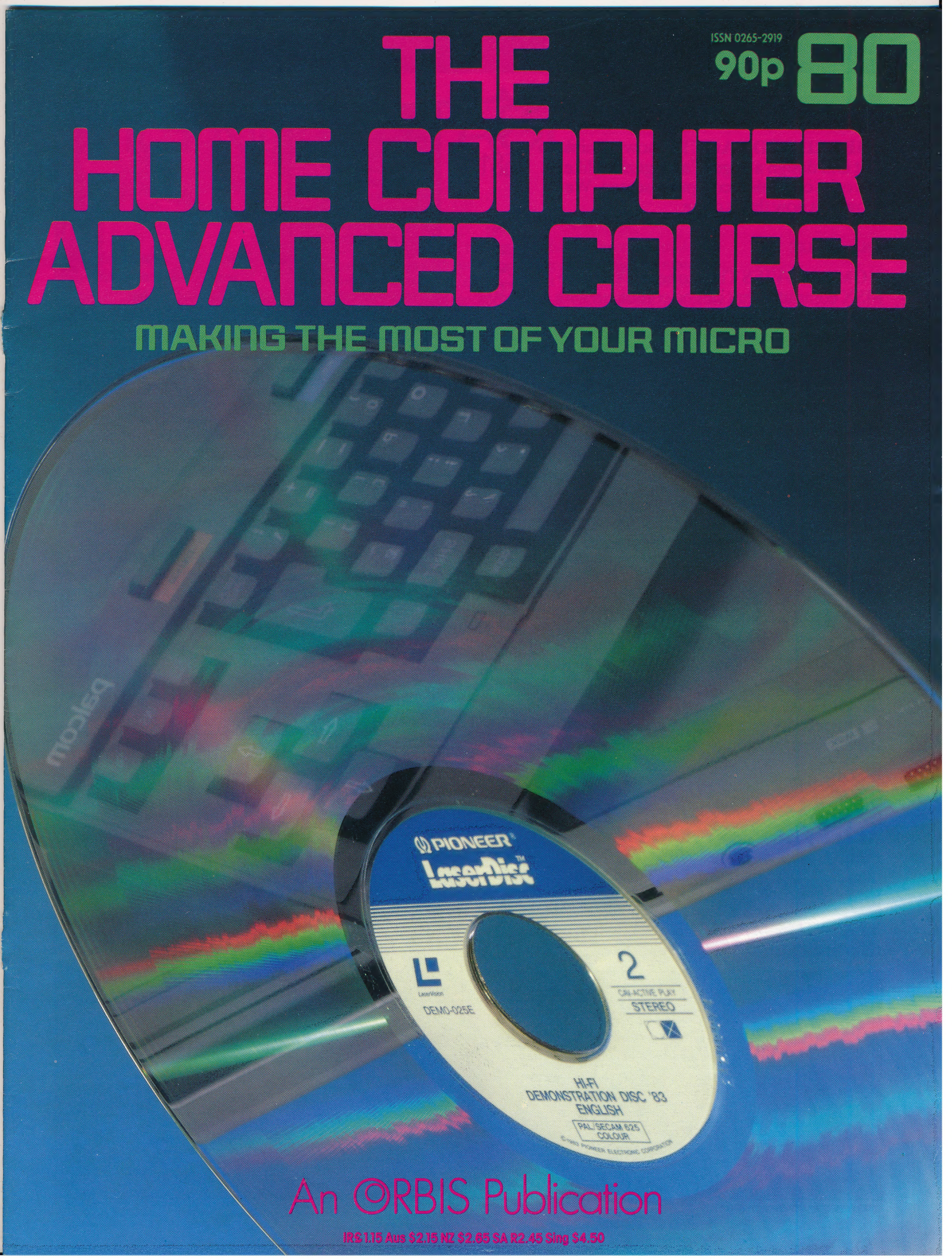
ISSN 0265-2919

90p 80

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



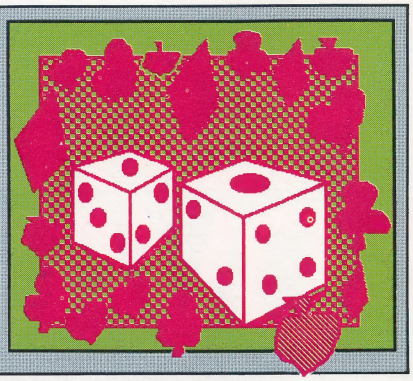
An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

Next Week

- We begin a short series investigating the use of microcomputers in the gambling industry.
- Following our general introduction to classical programming languages, we take a more detailed look at FORTRAN.
- We will design the decision trees for interaction and plot in our interactive character game.



QUIZ

1) The intersection of two sets is equivalent to which logical Boolean operation?
2) What is a 'kick', and where is its effect felt?
3) What design effect is indicated by the 'cropping tool' icon in the Apple Macintosh PageMaker program?

Answers To Last Week's Quiz

1) A direct entry system refers to a manuscript that is typed only once, and then manipulated by transferring the data through a compatible computer-based network.
2) The PC-DOS has been modified for the IBM PC/AT, preventing the original PC from reading the new disks.
3) You can either enter numerical data or a formula for manipulating data.
4) A semiconductor with a surplus of positive charges.

Amendments To The Interactive Character Game Listings

The following alterations should be made to the listings printed on page 1507:

```
80  GOSUB 2350: GOTO 100

2350 PRINT: RESTORE: FOR c=1 TO 7: READ
c$(c,1): GOSUB 4070: PRINT c$(c,1); "—";:
PRINT "Set this char?": GOSUB 4110
```


Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editors** Jon Kaye; **Contributors** Chris Honey, Chris Laing, Dougie Bern, Max Phillips, Mike Curtis, Steve Cooke, Steve Colwill, Steve Malone, Steven Vickers, David Mudd; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £..95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CANS1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS

THE PRINTED PAGE

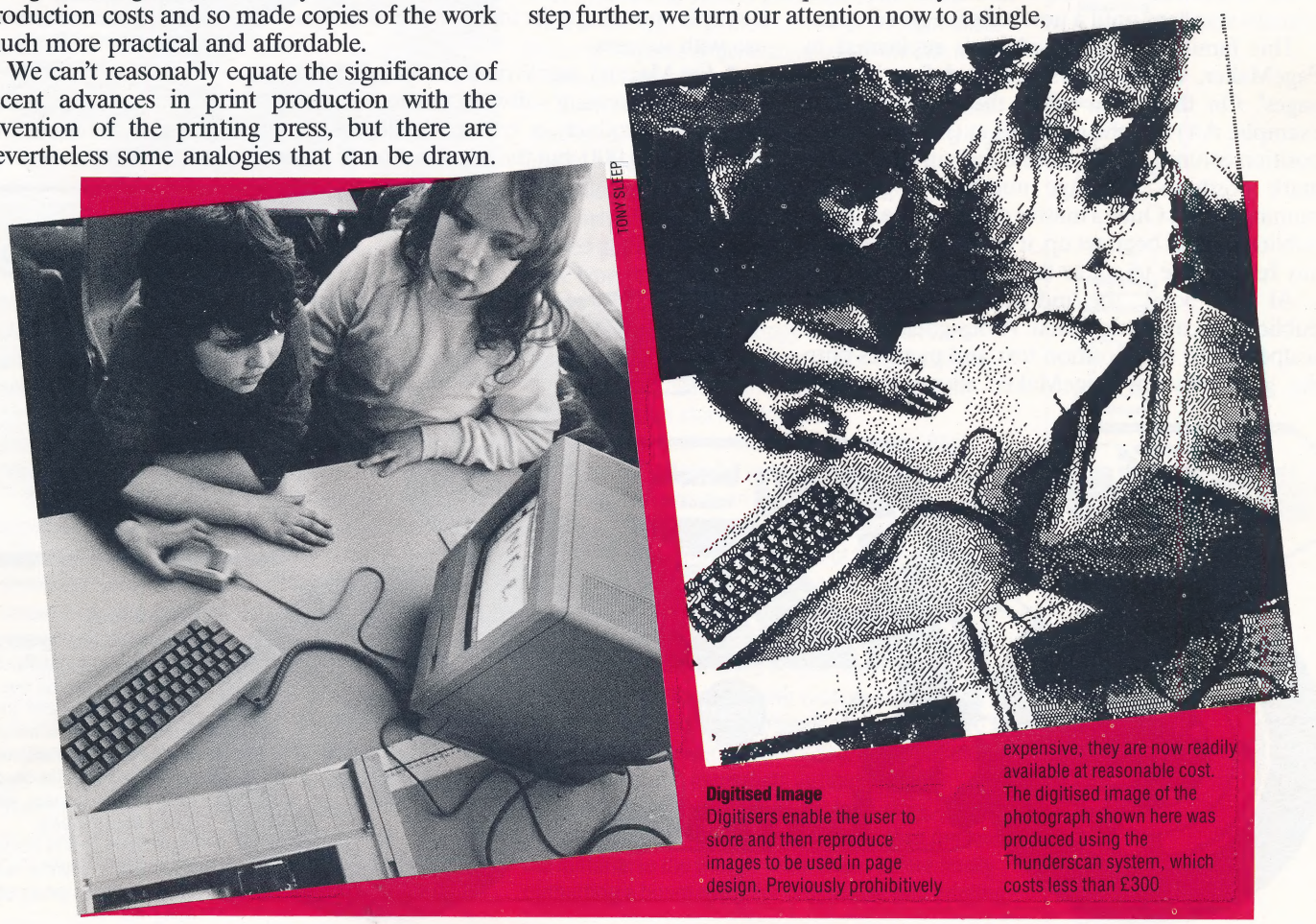
We conclude our short series on the introduction of new technology in publishing by looking at the reduction in production costs that can be achieved using a microcomputer-based layout and design system. We also demonstrate how such a system works using the PageMaker package for the Apple Macintosh.

The introduction of the printing press to Britain by William Caxton in 1476 was such an important event because it made the written word accessible to a much greater number of people. Before this, of course, the only books available were laboriously produced by hand, either written by scribes or made from blocks of wood into which individual letters had been carved. This meant that there were very few copies of any given volume available and the expense of producing them was prohibitively high. Johan Gutenberg's movable type printing press technique, which Caxton brought to England, drastically reduced the unit production costs and so made copies of the work much more practical and affordable.

We can't reasonably equate the significance of recent advances in print production with the invention of the printing press, but there are nevertheless some analogies that can be drawn. Primarily, the new production methods that rely on computer technology can produce material similar in quality to that produced by traditional printing methods, but at a much reduced cost. But while Caxton's initiative made books more widely available, it is likely that the computerised production process will have a different, though by no means less important, effect.

The cost of magazines and books is relatively low and so it is unlikely that reducing the price of printing material will lead to significantly increased sales. But for publishing companies, reducing production costs means being able to sell fewer copies to recoup its production expenses. This in turn may lead to an increased number of specialist publications catering for smaller audiences, which were previously not economically viable.

In the previous instalment (see page 1561), we outlined the general processes involved in producing a magazine and looked at some of the changes currently taking place in this area. Taking the idea of a streamlined production system one step further, we turn our attention now to a single,



TONY SLEEP

Digitised Image
Digitisers enable the user to store and then reproduce images to be used in page design. Previously prohibitively expensive, they are now readily available at reasonable cost. The digitised image of the photograph shown here was produced using the Thunderscan system, which costs less than £300

stand-alone microcomputer that can be used as word processor, layout designer and controller of a high-quality printer.

The Apple Macintosh, equipped with 512 Kbytes of memory (the 'Fat' Mac as it is known) has had several sophisticated pieces of software developed for it with this very purpose in mind. Included in these is a package called PageMaker; if a high-quality printer is added to this package, it becomes possible for a small group of people to produce near typeset-quality publications significantly faster and cheaper than by traditional methods.

HOW PAGEMAKER WORKS

The Macintosh, with its windows, icons and mouse approach, is an ideal environment for applications like PageMaker. The familiar pull-down menus and screen windows are present when PageMaker is loaded and the Mac's screen is set up as a desktop complete with tool box for the page designer to work on.

Normally, a designer works by starting with a layout grid, which is the standard page of a publication, and includes positional lines to assist in the placing of text columns, running heads, page numbers and so on. Copies of the grid can then be used to make up each page in the publication, ensuring that the publication has a uniform look. Traditionally, the designer's grid is drawn by hand when a publication is first launched and then remains standard until a new style is agreed upon.

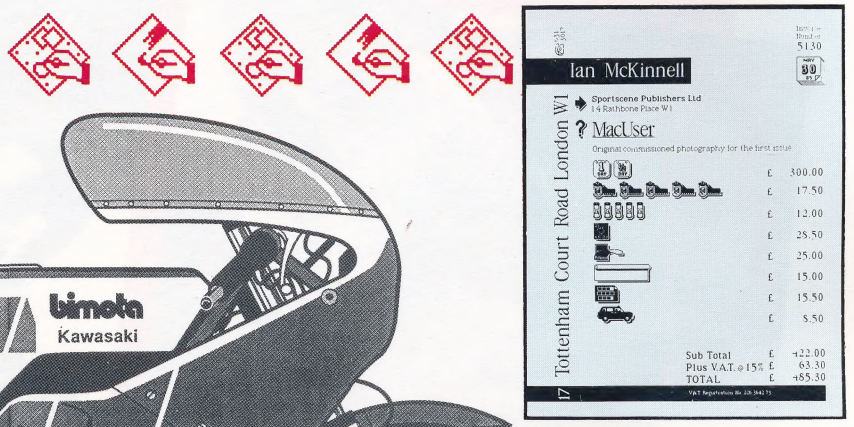
This familiar design method is replicated in PageMaker, which allows you to define 'master pages'. On these you select the page size (for example, A4) and the orientation (high or wide), position column guides, define margin widths and mark positions for page numbers, logos and running heads. Once a master page for a particular publication has been set up, it can be called back at any future time to act as the designer's grid.

At this point, the individual pages of the publication can be designed. A designer will use a scalpel to cut and position text and pictures onto the grid, whereas PageMaker allows text and graphics to be loaded from disk. And because it is compatible with other Mac packages, an article could be written with the MacWrite word processor, loaded directly into PageMaker and positioned on the board. Graphics designed on MacPaint and MacDraw, like a number of illustrations in this publication, can also be loaded by PageMaker and positioned on the page.

Once the various components that will go to make up the page have been loaded, you can use the various devices contained in the toolbox in the corner of the screen. These tools, for example, allow you to add or edit text on screen by selecting the A icon from the toolbox or, by selecting the 'cropping tool' icon, let you shrink, enlarge or trim graphics to fit into the space available. Using the mouse, text and graphics can be picked up and moved around the design grid at will and even 'laid down' on the desk surface temporarily while another job is being done.

PageMaker's toolbox also includes facilities that allow you to add special touches to the page, such as drawing borders around specific sections. It includes icons that control the drawing of lines, circles, ellipses and rectangular boxes with or without rounded corners. The thickness of the lines can also be selected by pulling down the appropriate menu. Once these shapes are drawn they can be filled with black or white, various shades of grey, or patterns. Although an incredibly useful tool for professional designers, PageMaker is simple enough for the inexperienced designer to use with success.

A Fat Mac, a LaserWriter (see page 1529) and the page processing software needed to set up a one-micro production system would cost in the region of £10,000, but the introduction of cheaper



Invoice Icons
This invoice was designed and printed using copy directly generated by an Apple Macintosh. Icon design is often an important feature of publications such as The Home Computer Advanced Course, making different sections readily identifiable. Similarly, company logos and product symbols are often used in invoices and letterheads

IAN McKINNELL

Portable Pixels
Files created by programs such as MacDraw and MacPaint can be stored on disk and then loaded into PageMaker and manipulated prior to being printed out. Both MacDraw and MacPaint provide the designer with powerful facilities, which can be used to create illustrations such as this (see page 1273 for a further example of an illustration generated using MacPaint)

WIMP-based machines, such as the Atari 520ST (see page 1549), which incorporates the GEM environment, is likely to result in dramatic price reductions for such systems. Even at current prices the time and attendant labour costs saved by producing a publication using computerised page design systems significantly undercut conventional production costs.

Cutting Costs

How much time and money would actually be saved using an electronic page processing system, such as the Macintosh with LaserWriter and PageMaker? Let's envision the Mac system being used to produce a 16-page professional-quality company newsletter. Assuming that the publication is produced by a small team of editorial and design staff employed by the company, and that all work except the typesetting is done 'in-house', the following represents a breakdown of typical production costs:

Job	Rate	Cost
Marking up copy for typesetting	5 hours @ £5 per hour	£25
Cost of typesetting and corrections		£250
Co-ordination with typesetters	5 hours @ £5 per hour	£25
Proofreading	8 hours @ £5 per hour	£40
Layout and design	6 hours @ £5 per hour	£30
Final paste-up	16 hours @ £5 per hour	£80
	Total Production Costs	**£450**

The equivalent publication could be designed with the PageMaker by one person in about five hours at, say, £10 per hour. The 16 pages could therefore be produced for around £50. But there are other, associated benefits. The conventional method is, of necessity, sequential, and the various tasks need to be carefully scheduled, which, in our example, would take around two weeks. Using PageMaker, much of the organisational problems of producing a publication disappear, because text corrections, cuts and additions, and captions for illustrations can all be added on-screen during the design process.

Setting The Pace

Modern litho printing techniques use photographic plates to transfer an image onto paper. Pages prepared for press must therefore be 'camera-ready', and where text is concerned, this involves the production of a perfect image of the text to be printed, as opposed to the raised type that might be produced by a hot-metal composer.

Phototypesetting machines do exactly this, accepting text as input from a keyboard or disk and outputting it in the form of 'galleys' — long strips of photographic paper onto which the text has (hopefully) been perfectly printed and ready to be pasted onto the board.

Early phototypesetters used a spinning-disc-shaped template that held a copy of all the characters in a particular fount. Photosensitive paper passed on one side of the disc, and as the desired letter spun into position, a light would flash, transferring the character onto the paper. Although relatively primitive, these systems reached considerable degrees of sophistication and were able to generate text reasonably quickly.

The photograph shows a modern laser-based phototypesetting system produced by Linotype-Paul, and is typical of modern computer-controlled systems, capable of generating text at the rate of up to 368,000 characters per hour.

There are three main components to the system illustrated. On the left is the Linotronic 300 — a high-quality wide-measure laser typesetter with a resolution of up to one million pixels per square centimetre. Type can be expanded or condensed, slanted and reversed as desired, and there can be up to 32 typefaces on line.

The terminal in the centre enables text to be entered manually, from floppy disk, or even via a modem. This is useful for journalists, for example, who very often need to enter text into the system while covering a story on location.

The display unit on the right, a Linotype-Paul Typeview 300, displays text in the style, size and position it will occupy in the finished setting. Systems like this not only enable camera-ready copy to be produced with great ease, but they can also be linked to larger network systems that can be accessed by all kinds of publishing companies. Linotype, for instance, offers a system capable of handling up to 40 interactive terminals simultaneously together with vast on-line storage and shared access to peripherals. A single system employed by a newspaper, let's say, could therefore accept input from journalists in-house and on location, retrieve feature material from store, pass it to editors, combine it with artwork and then present the material to the design department before finally outputting a board ready for processing and printing

VERSATILE MACHINE

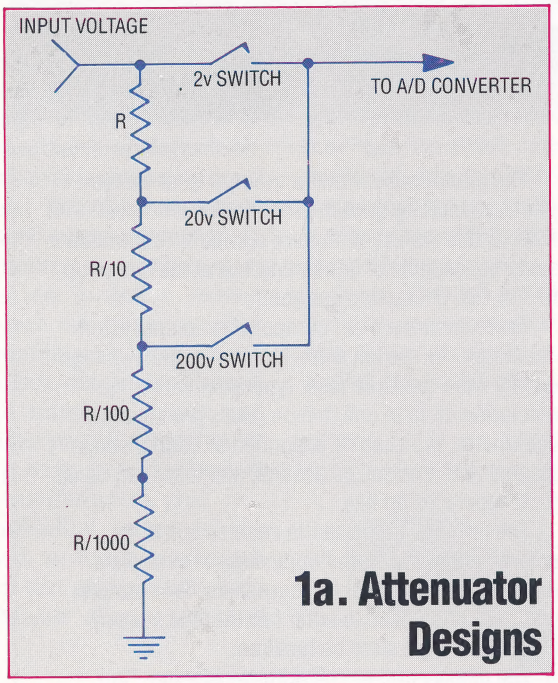
In this series we have examined the theory behind the workings of A/D converter chips, developed a practical circuit for building a digital multimeter and shown you how to assemble it. In this penultimate instalment, we begin to look at add-ons that can increase the meter's versatility.

As it stands, the digital multimeter can measure voltages from 0.0000v to 1.9999v, which is why we say it has a basic sensitivity of 2v. By attenuating the signal to be measured, it would theoretically be possible to measure voltages up to 19,999v (20 Kvolts). Such high voltages can be extremely dangerous so we've limited the upper range to 199.99v

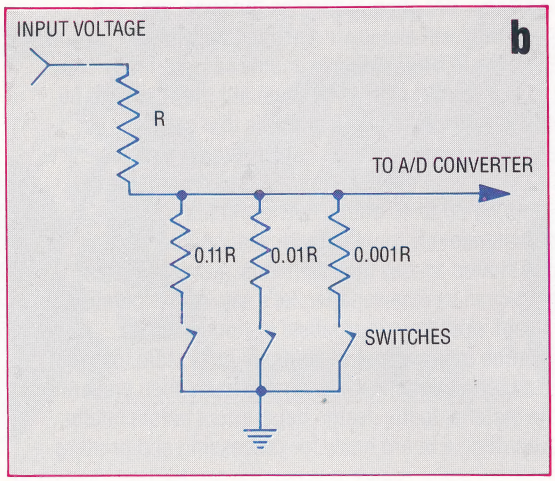
You may have noticed from the diagram in the last instalment (see page 1572) that there were three wires left unconnected from three of the LEDs. These are connected to the decimal points that appear to the right of the digit on the LEDs and are used to switch the decimal point on when required. A one-pole three-way switch 'ganged' to the input attenuator switch makes sure that the appropriate decimal point is on for the sensitivity selected. First let's consider the input attenuator circuit itself.

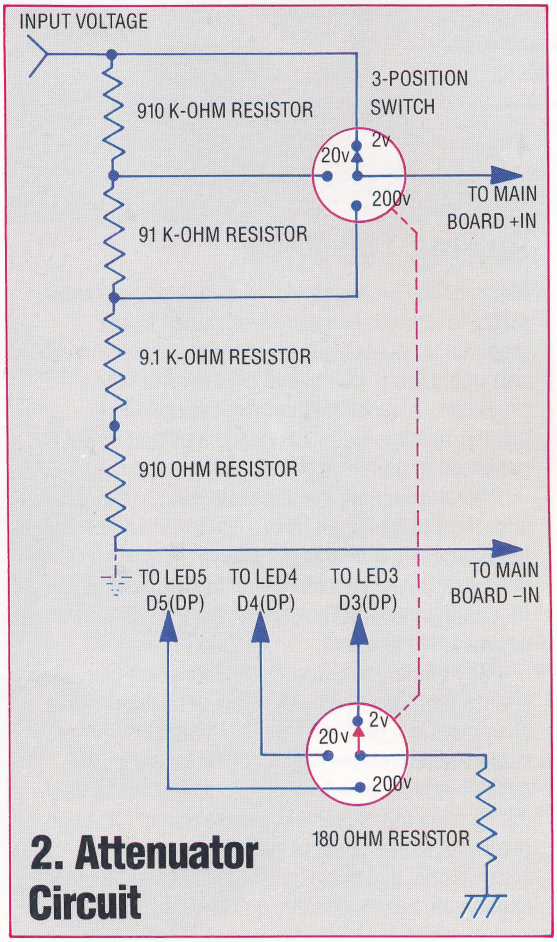
Basically, there are two ways of making an input attenuator. Both of them involve the construction of a potential divider network that spreads the input voltage over a number of resistors, with a fraction of the input voltage being tapped off for measurement as appropriate. These two types of attenuator are shown in diagram 1. The first type has the advantage of simplicity, as well as being readily adaptable to the measurement of current and resistance. This is not true for the second

Attenuator Designs
To increase the range of our basic voltmeter we need to design an input attenuator that will scale the voltage down to a level that is acceptable to the main A/D conversion circuit. Both use a number of resistors that can be switched into the circuit to tap a proportion of the input potential. We will use the first of the two designs shown here, because the second design has the disadvantage of requiring a special input protection circuit (although it can incorporate solid-state analogue switches)



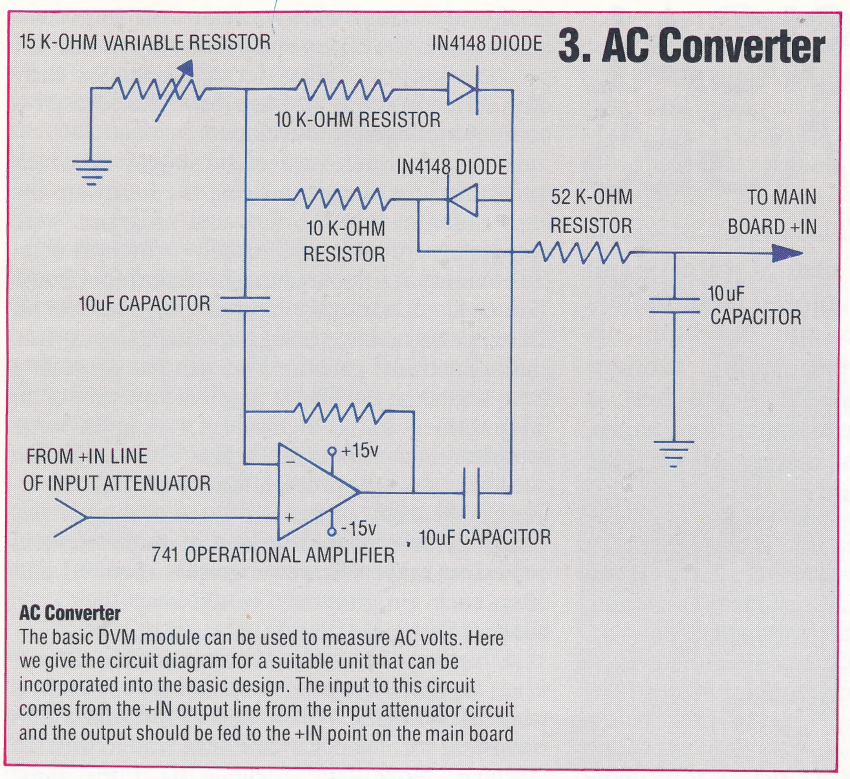
1a. Attenuator Designs





2. Attenuator Circuit

Attenuator Circuit
Use a small piece of 0.1in matrix board to construct the input attenuator. The resistors should have 1% tolerance on the values stated. Mount the resistors in a line along the board and tap off at the ends of the 910 K-ohm and 91K-ohm resistors to a three-pole double-throw switch. The positive and negative outputs should be tied to the main circuit board (see page 1553) with short lengths of covered wire. The three remaining tags on the double-throw switch should be connected to the decimal point pins on LEDs 3, 4 and 5 of the display (see page 1572). These connections enable the decimal point in the display to be automatically repositioned when a new range is selected



AC Converter
The basic DVM module can be used to measure AC volts. Here we give the circuit diagram for a suitable unit that can be incorporated into the basic design. The input to this circuit comes from the +IN output line from the input attenuator circuit and the output should be fed to the +IN point on the main board

attenuator. In an auto-ranging circuit (where the input attenuation is switched automatically), the second type has the advantage of being able to use state analogue switches, whereas the first can only use mechanical switches. We'll be using the second type in our design because of the ease with which it can be adapted to make measurements.

The actual circuit (shown in diagram 2), together with the decimal point switching, can easily be constructed on a small piece of matrix board, substituting and connecting the components specified in the parts list for the items in the circuit diagram. The resistors specified are metal film types with one per cent tolerance. Layout is not critical, but it is advisable to keep all lead lengths short as the input impedance of the A/D chip is very high and long leads act like an antenna for spurious signals.

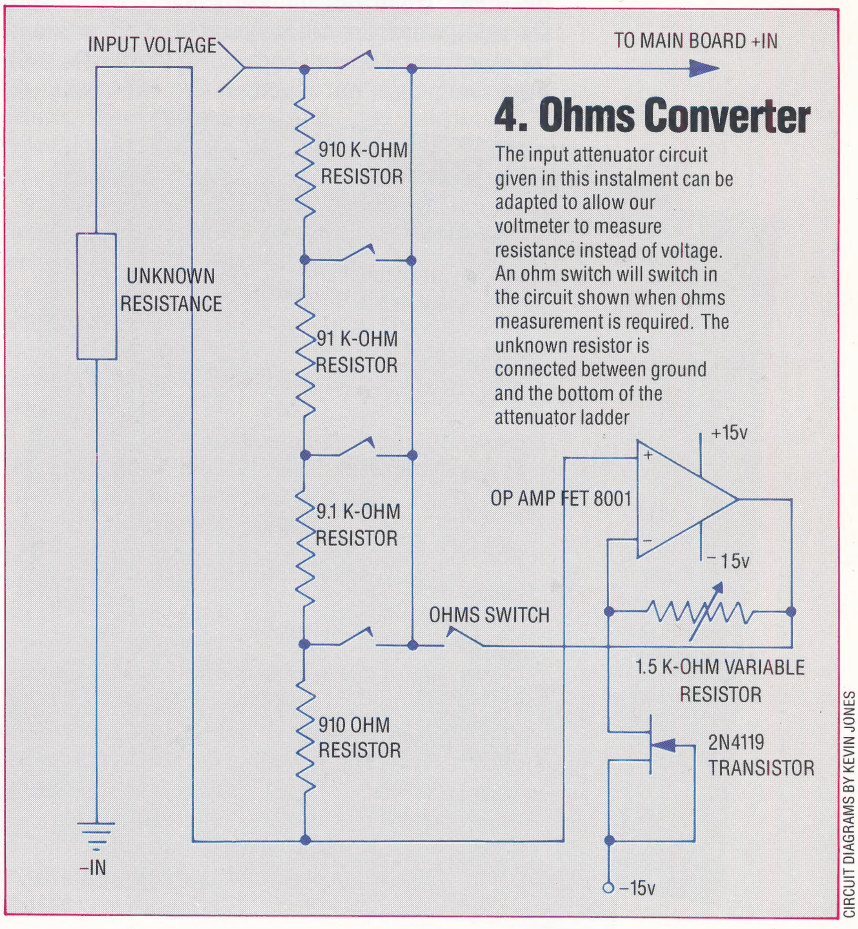
This attenuator network is simple in design and uses readily available resistor values. It has the disadvantage, however, of presenting a relatively low load to the circuit being tested (just over 1M-ohm) but this isn't a serious limitation — it works out at a drain of about 5uA if a 5v signal is being measured.

The circuit we've presented can measure DC volts in three ranges — 2v, 20v and 200v. Relatively simple changes or additions to the basic circuit will allow many other units to be measured, including ohms, AC volts and temperature. We provide a number of possible additions to the basic circuit, but these are in circuit diagram form only.

To measure AC volts using a DC voltmeter (such as our DVM), you only need to rectify the AC signal to DC and measure it. Unfortunately, a simple diode rectifier will not do and what is known as a precision rectifier is required. One possible circuit is shown in diagram 3. It is based on the well-known and very low-cost 741 integrated operational amplifier. The only disadvantage here is the need for a ± 15v power supply, but this could be derived from the 12v AC output of the mains transformer; the current requirements of the 741 are very low.

Measuring ohms on a conventional analogue voltmeter is easy, since the voltage dropped across a resistor being measured is proportional to its resistance. It is not so easy to do with a digital voltmeter, however, and so an ohms converter circuit is required. The ohms converter (see diagram 4) works by applying a constant voltage to the input attenuator ladder, which generates a constant current across the resistor being measured. Since a constant current across an unknown resistor will drop a voltage across it that is proportional to the resistance, this voltage drop can be read directly by the DVM. Again, an operational amplifier will be required, and it will need to be an ultra-high input impedance type.

By applying the output of the ohms converter to the attenuator ladder, four ranges of ohms can be measured: 2 K-ohm, 20 K-ohm, 200 K-ohm and 2 M-ohm. (That's why two separate resistors were used at the bottom of the ladder instead of the single 10 K-ohm resistors that would have been adequate if only three voltage ranges had been required.) Naturally, a four-way attenuator switch would be required instead of the three-way switch specified.



CIRCUIT DIAGRAMS BY KEVIN JONES

CLASSICAL LANGUAGES

While early languages such as FORTRAN and COBOL may seem far removed from today's micros, their concepts and developments have much bearing on modern programming theories. We begin here a series devoted to these early languages, starting with a general discussion of their history and development.

To most of us the idea of 'programming' a computer is fairly commonplace. Even if we cannot do it very well, we at least know something of what it is all about. Things were very different during the late 1940s and early 1950s, however, when the concept of a programming language was first developed. This early period produced a number of languages which are still in use today.

The object of this series of articles is to examine how these languages have developed into their present form, as well as tracing some of their influences on more modern languages and discovering their relevance to modern microcomputers. We will take a close look at two of these languages in particular, COBOL and FORTRAN, which are not only still used but account for more actual lines of code being written than all other languages put together. We will also look briefly at ALGOL, which remains an important language but has been superseded by PASCAL for practical purposes. First, let's consider the early days of computing and see how the idea of a computer 'language' developed.

If we omit the very early stages, in which 'programming' meant rewiring the hardware for each different job, the first real programs were completely numeric and usually written in octal (base eight) as a convenient shorthand for binary. These were laboriously entered using switches on a front panel or, later, using paper tape or punched cards. Each new program was written completely from scratch at the lowest possible level and often programmers found themselves writing the same routines time and time again. From this, the concept of a subroutine library developed — though the term 'subroutine' was not then in use. These 'libraries' would be kept in notebooks and copied out into each new program as required, and often each 'programmer' would keep their own libraries with only occasional sharing.

A big step forward was taken in Manchester when the EDSAC system was built in 1951 and the work of Wheeler, Wilkes and Gill resulted in a consistent set of general subroutines to go with the machine. Now that everyone used the same subroutines, programs became easier to write and began to show structural resemblances to their modern counterparts. Blocks of code performed the particular task with subroutine calls to carry out the standard functions such as input, output and numerical calculation, which were common to most programs.

When the memory capacity of the machines increased, it became possible to store the subroutine libraries internally, or at least on-line, and from here it was only a short step towards the development of a code that allowed the programmer to specify the subroutines required using alphabetic characters and a mathematical notation. A program would look at each line of this code, determine the subroutines required, call them, and then go back to the next line. This is essentially the way that a modern BASIC interpreter works. An example of this was the 'Short Code' produced by John Maunchly in 1949 on the BINAC computer and later the UNIVAC.

A further refinement of this early technique took each subroutine as required, but instead of performing it directly, it was first copied to an output device, thus ending up with a complete executable program. It was at this stage that the first compilers were developed, with the term 'compile' referring to the way in which the program was put together from a number of components arranged into a logical order, in the same way as you'd compile a set of essays.

EARLY COMPILERS

One of the earliest successful compilers was the A-2, developed by a team led by Grace Hopper at Remington Rand in 1955. This used the 'three address' concept, in which each operation had a

Landmark Dates

A simple chronology of the early development of programming languages:

1951 First stored program digital computer, EDSAC, completed at Manchester

1952 John Maunchly's Short Code

1953 Laning and Zierly's Speedcoding

1954 A2 compiler
First specifications for FORTRAN

1955 First actual compilers available for FORTRAN and FLOW-MATIC

1956 General release of FORTRAN and FLOW-MATIC compilers
Specification for FORTRAN II

1957 Specification and release of first ALGOL compiler

1958 Release of FORTRAN II

1959 First specifications for COBOL
First compiler for LISP

1960 Release of first COBOL compiler
Release of revised ALGOL 60

mnemonic name followed by three addresses — two for source data and one for destination. In some respects, it was very similiar to an assembler except that the instructions didn't fit onto any specific machine architecture. This language was later developed into ARITH-MATIC, which was followed by a similar language from Remington Rand — AT3, or MATH-MATIC.

By about the end of 1951, a number of people had realised that from the point of view of the outside user, the computers were in effect running programs that had been written using a different code to the native machine code. The fact that the computer was doing a translation job first was irrelevant.

In this case, you might as well design your new language in order to make the programmers' task of writing easier, rather than to make the job of translation easier, especially since the hardware was getting faster and bigger, and the programmers couldn't keep up.

The next problem to be tackled was that there was no form of standardisation, because each machine used its own language geared to a particular small set of problems. The next step was therefore the development of a language independent of any particular hardware specifications to cater for a wider class of problems. This approach was begun in 1954 and led to FORTRAN (IBM mathematical FORmula TRANslation system), the first real programming language.

FORTRAN is a mathematically based language, well suited to the largely numerical work that was currently practiced and similar to many of the autocodes around at the time. It was, however, still based to a large extent on the machine architectures available. The business community was becoming interested in the possibilities of using computers for large scale data processing, but they needed a language that more closely resembled common business English.

It's amusing to think that at this time there were a number of programmers who thought this impossible as there was no conceivable way to make the computer 'understand' words instead of numbers. They were soon proved wrong, however, by (among others) Grace Hopper, who developed a language called FLOW-MATIC, which could be read and understood by management as well as by programmers. By 1956, this had developed into COBOL (COmmon Business Oriented Language). These languages had rigid specifications.

LANGUAGE THEORY

By this time, the numbers of computers and programmers had increased dramatically and people began to look more closely at the theoretical aspect of programming languages, trying to find the most efficient and elegant way of expressing algorithms. This led to the development of ALGOL (ALGOrithmic Language) in 1958. ALGOL never achieved the widespread popularity of FORTRAN or COBOL, but it nevertheless occupies an important place in the development of languages. It was ALGOL that first embodied the principle of sound program design, which has been a major consideration in all subsequent languages.

A number of other languages were developed in those early days, many of which still remain in use. One of the best examples is LISP (LISt Processing language) which was developed in the period 1956 to 1958. It's not only still in use but of increasing importance in the field of artificial intelligence. The three languages, FORTRAN, COBOL and ALGOL, however, have represented the mainstream of programming over the last two decades. They have all undergone a number of revisions over the years to incorporate new features and to reflect modern trends in language design. The current revisions are FORTRAN 77 (specification issued in 1977), COBOL 74 and ALGOL 68.

Short Code

In John Maunchly's Short Code, the familiar BASIC assignment:

```
10   A = B + C
```

would be written as

```
10 S0 03 S1 07 S2
```

where 10 is the line number and S0, S1 and S2 are symbols representing the 'variables' A, B and C as single words of memory. 03 and 07 are the operation codes for assignment and addition, respectively.

In the A2 language, the same statement would appear as:

```
ADD B C A
```

In 1964, a much simplified version of FORTRAN (with some influences from ALGOL) was introduced at Dartmouth College in the US to make the task of learning to program much simpler, and to use the new time-sharing multi-user systems that were becoming available. This became known as BASIC (Beginners All-purpose Symbolic Instruction Code).

When the new 1968 standard for ALGOL was being prepared, Niklaus Wirth disagreed with the way in which the language was being made more complex and argued in favour of a simpler and more elegant approach. ALGOL 68 is generally believed to be too complex for normal use and is rarely found outside the highest reaches of academia. However, Wirth's much simplified version, PASCAL, has attained great popularity.

COBOL has not spawned derivatives in the same way, since, in its own field, it has been used almost exclusively. The enormous volume of code that is written in COBOL has meant that it has been used as a vehicle for a number of developments in the area of program and system design; for example, program generation, structured program design and the use of databases.

Box Office

Simple seat reservation systems:

FORTRAN IV:

```
C       THEATRE SEAT RESERVATIONS.
C       PROGRAM TO ACCEPT A SEAT NUMBER,
        CHECK IF ALREADY BOOKED,
C       BOOK IT IF AVAILABLE OR GIVE
        MESSAGE IF ALREADY BOOKED.
C
C       DECLARE VARIABLES
C
        INTEGER SEATNO, SEAT (500)
C
C       MARK ALL SEATS AVAILABLE
C
        DO 100 I = 1, 500
100     SEAT (I)=0
C
C       READ SEAT NUMBER AND CHECK
        AVAILABILITY
C
101     READ (1, 10) SEATNO
        IF (SEAT (SEATNO). NE. 0) GOTO 102
C
C       SEAT IS AVAILABLE
C
        SEAT (SEATNO)=1
        WRITE (1,20)
        GOTO 103
C
C       SEAT IS NOT AVAILABLE
C
102     WRITE (1,30)
C
C       NEXT SEAT NUMBER
C
103     GOTO 100
C
C       FORMAT STATEMENTS
C
10      FORMAT (I4)
20      FORMAT (1H, 17HSEAT IS AVAILABLE)
30      FORMAT (1H, 19HSEAT ALREADY
        BOOKED)
        END
```

ALGOL 60:

```
comment THEATRE SEAT RESERVATIONS.
PROGRAM TO ACCEPT A SEAT NUMBER, CHECK IF
ALREADY BOOKED, BOOK IT IF AVAILABLE OR GIVE
MESSAGE IF ALREADY BOOKED
NOTE THAT ALGOL DOES NOT INCLUDE INPUT/
OUTPUT STATEMENTS:
  begin
    integer SEATNO, I;
    integer array SEAT[1:500];
    comment MARK ALL SEATS AVAILABLE
    for I:=1 step 1 until 500 do
      SEAT[I]:=0
    comment READ SEAT NUMBER AND CHECK
    AVAILABILITY;
  NEWSEAT:((read SEATNO));
    if SEAT [SEATNO] = 0 then
    begin
        SEAT [SEATNO] := 1;
        ((print 'SEAT IS AVAILABLE'));
      end
      else
        ((print 'SEAT ALREADY
        BOOKED'));
      goto NEWSEAT;
  end
```

COBOL 74 [DATA and PROCEDURE division only] :

```
*THEATRE SEAT RESERVATIONS.
*PROGRAM TO ACCEPT A SEAT NUMBER, CHECK
IF ALREAD / BOOKED,
*BOOK IT IF AVAILABLE OR GIVE MESSAGE IF
ALREADY BOOKED.
DATA DIVISION.
WORKING STORAGE SECTION.
01 SEAT-AVAILABILITY.
   02 SEAT PIC 9 OCCURS 500 TIMES.
77 SEAT-NUMBER PIC 999.
77 LOOP-COUNTER PIC 999.
77 SEAT-AVAILABLE PIC X (17) VALUE
   'SEAT IS AVAILABLE'.
77 SEAT-BOOKED PIC X (19) VALUE
   'SEAT ALREADY BOOKED'.
PROCEDURE DIVISION.
MAIN PARAGRAPH.
*MARK ALL SEATS AVAILABLE
   PERFORM MARK-SEAT-AVAILABLE-PARAGRAPH
      VARYING LOOP-COUNTER FROM 1 BY 1
      UNTIL I > 500.
* READ A SEAT NUMBER AND CHECK AVAILABILITY
   PERFORM GET-SEAT-NUMBER-PARAGRAPH.
   STOP RUN.
MARK-SEAT-AVAILABLE-PARAGRAPH.
   MOVE ZERO TO SEAT (LOOP-COUNTER).
GET-SEAT-NUMBER-PARAGRAPH.
   ACCEPT SEAT-NUMBER.
   IF SEAT (SEAT-NUMBER)IS EQUAL TO 0
      MOVE 1 TO SEAT (SEAT-NUMBER)
      DISPLAY SEAT-AVAILABLE
   ELSE
      DISPLAY SEAT-BOOKED.
   GOTO GET-SEAT-NUMBER-PARAGRAPH.
```

Seat Reservations
We show here listings in three different high-level languages — FORTRAN, ALGOL and COBOL — demonstrating program structure and some comparative features. Each program accepts a seat number as input, checks to see if the number has previously been entered ('booked') and, if not, marks that seat as taken

BBC HULTON PICTURE LIBRARY

HOME MOVIES

Unlike most MSX-standard micros, the Pioneer PX-7 personal computer features a wide range of interfaces. These allow it to control the Pioneer LD-700 and LD-1100 laser disc players and the SD-26 television system — making it much closer to the original MSX concept of a complete home entertainment system.

We have already looked at the way laser discs controlled by computers offer new possibilities in the fields of education and home entertainment (see page 201). Pioneer's PX-7 personal computer, linked to the LD-700 (as shown here) or the more sophisticated LD-1100 laser disc player, offers a realisation of those possibilities at a price within reach of the home user. The PX-7 is a highly developed MSX home computer, while the LD-700 is a budget-priced laser disc player that uses Phillips-style 12in laser discs.

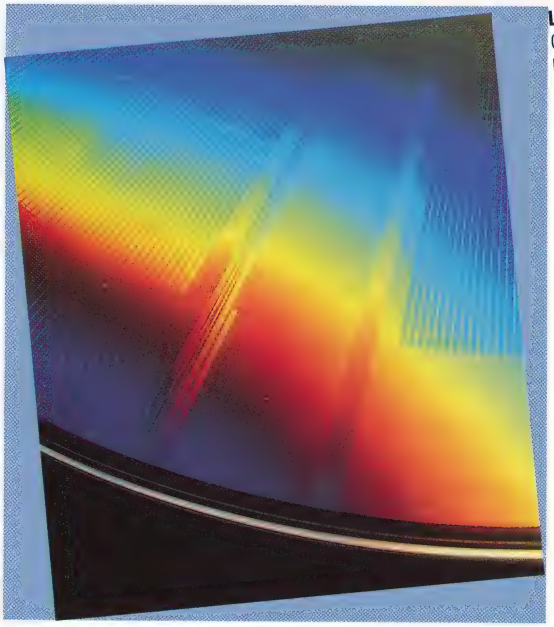
The PX-7 is very unusual for an MSX computer, being the first to have a detached keyboard. This allows the CPU unit to be stacked near or on a video recorder, laser disc player, television set or hi-fi system. Accordingly, the CPU unit is designed to look more like a hi-fi component rather than a home computer, and the keyboard is connected via a generous five-foot lead. On the front of the machine is an overall volume control, a socket for headphones and a mixing control, which adjusts the balance between sound generated by the computer and that coming from an external source such as the laser disc. An audio-video through switch effectively cuts off the computer allowing external video and audio signals to pass straight through to a hi-fi and television connected to the unit.

Inside, however, is a standard 32 Kbyte MSX computer, which can be used with the full range of MSX software and peripherals now available. Surprisingly, despite the sophisticated laser disc technology, the PX-7 uses cassettes to store its programs and information. For those that do wish to delve deeper into the micro's potential, a second cartridge connector on the rear will accommodate disk drives and other add-ons.

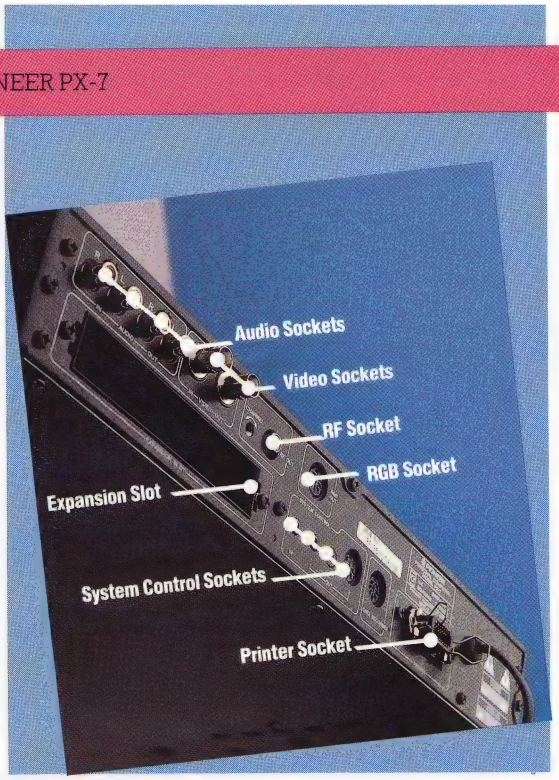
There's a complete range of computer interfaces for television, composite or RGB monitor, cassette, twin joysticks, two cartridges and Centronics printer. In addition, the unit has stereo connections for a hi-fi and a 'system control' interface. The latter is a general purpose interface and can be used to connect to laser disc players, video cassette recorders and so on. As new equipment becomes available, the interface

Social Interaction
Pioneer's PX-7 MSX home computer and LD-700 laser disc player are designed to work together. The PX-7 can be programmed using extensions to standard MSX BASIC to control the laser disc player so that individual frames or sequences can be selected and displayed. Computer graphics and sound can also be mixed with the video output from the disc player to produce interactive video under computer control

CHRIS STEVENS



Laser Tracks
CAV (constant angular velocity) laser discs, like magnetic discs, offer direct access to any part of the disc. The controlling mechanism can locate individual sections on the disc's surface by referring to the radial tracks shown here. Each ring on the disc corresponds to a frame of the programme stored, the corresponding part of the radial track holding the frame number



Sockets Galore
One of the outstanding features of the PX-7 is its comprehensive range of sockets, which allow connection via standard cables to a complete range of audio and video equipment

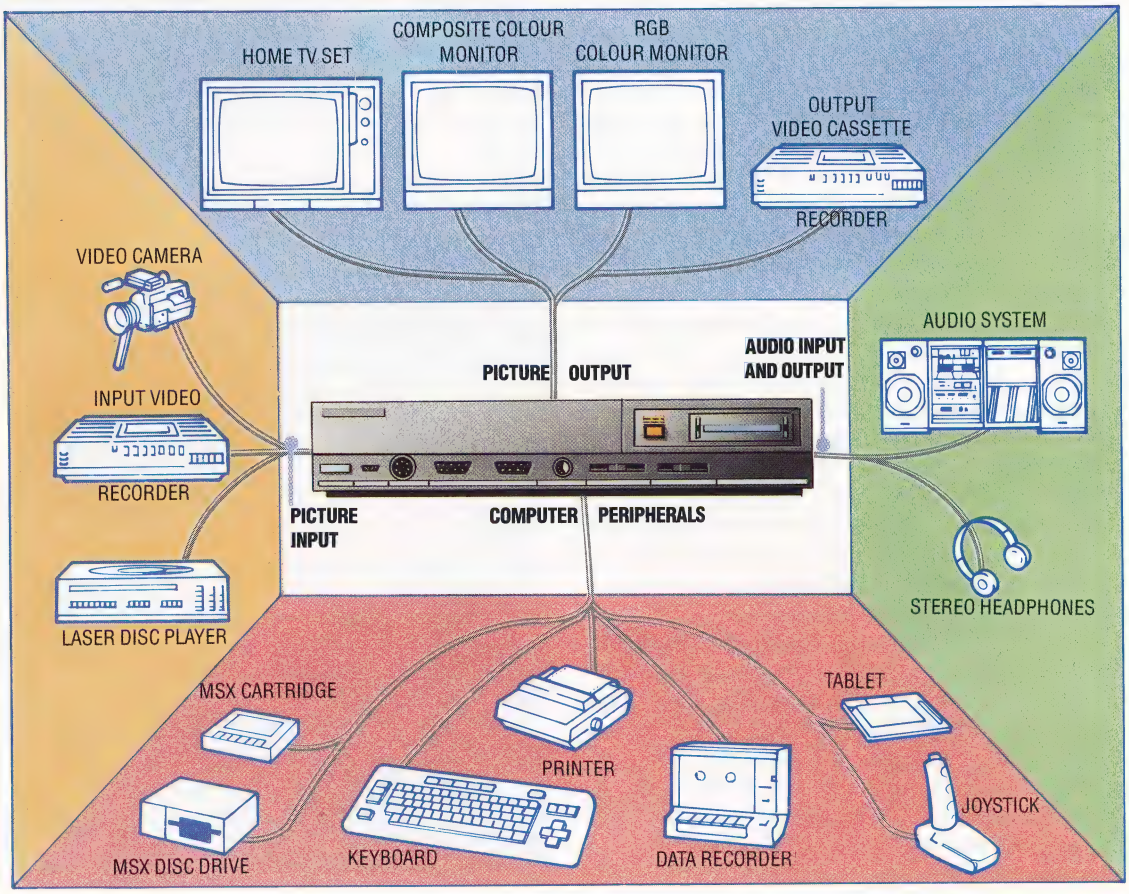
and its controlling software will allow it to be added to the system. Several devices can be connected at once, each having a unique 'device code' to allow a program to specify the device for which an outgoing command is intended.

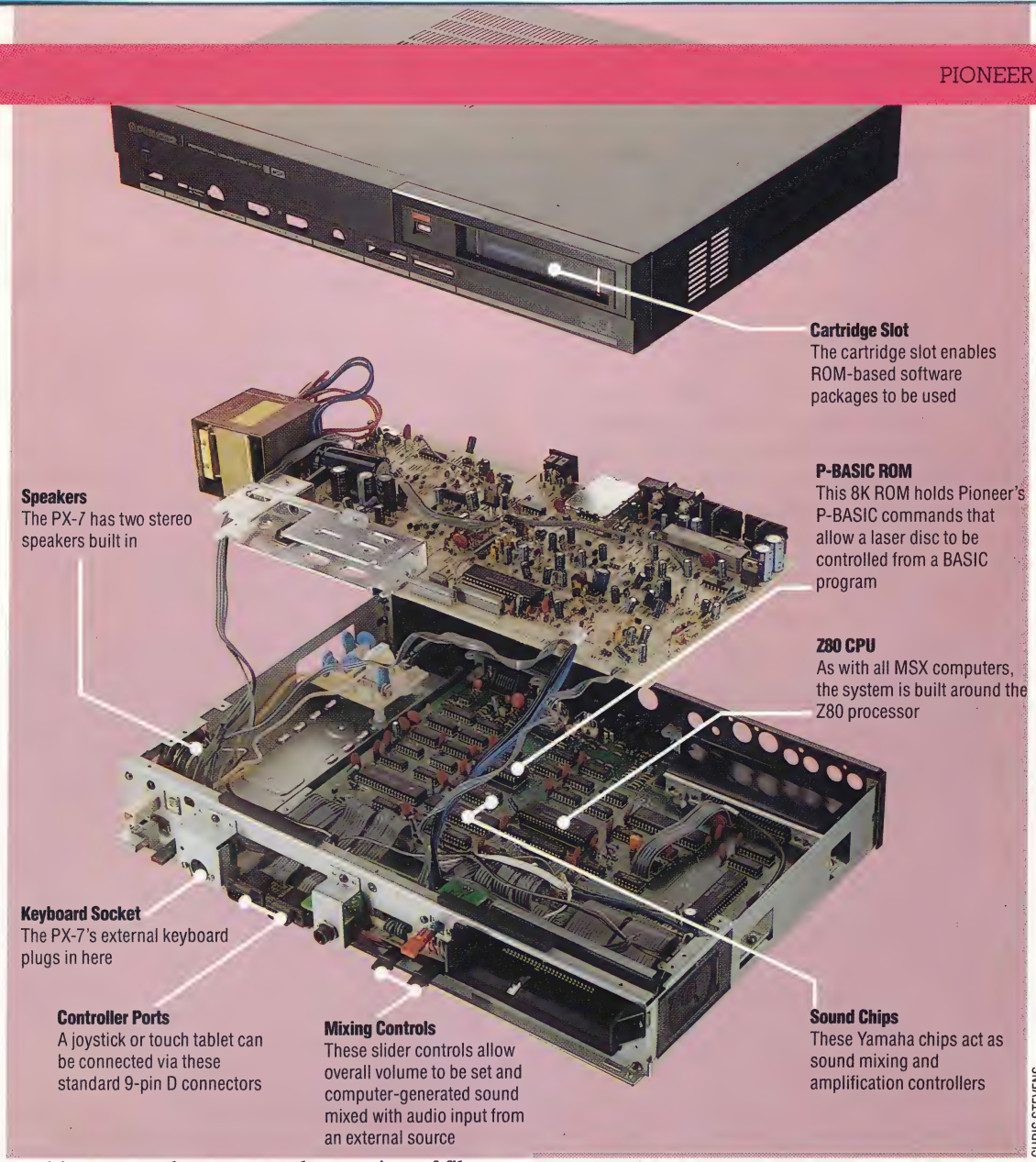
Controlling the interface is made easy by a set of extensions to the MSX BASIC held in ROM. When the computer is first switched on, you can choose to run ordinary MSX BASIC or P-BASIC (MSX BASIC with system control commands). The new commands take the form of CALL statements, so that MSX BASIC itself remains standard. Most of the 16 new commands deal with controlling the video and audio connections of the computer. It is possible to show the computer's display, the incoming video display (such as a laser disc) or superimpose the two. This allows computer graphics and text to appear over the top of pictures coming from the laser disc or video tape. You can also flip between the computer, video and superimpose modes, using four extra keys on the keyboard — the only additions to an otherwise standard MSX layout.

The sound commands allow you to mute one or both stereo channels and adjust the balance between them. There's a set of extras to enhance MSX BASIC — commands that clear the screen in a variety of ways, save and load the display to a cassette unit and so on. However, the most important command is CALL REMOTE, which sends an instruction to a device connected to the system control interface.

There are a number of commands specifically for laser disc control. For example, CALL SEARCH (0,F,2000) will ask the player to search for frame 2000 on the disc. CALL FRAME is the most sophisticated command. Once executed, the subroutine specified will be automatically executed when the player shows a particular frame or chapter on the disc. This allows a program to be closely tied to the laser disc system. For example, in an educational program, the computer could ask Do you want to know more about this? whenever a student played a particular part of the disc, and

Mission Control
One of the original intentions of the MSX standard was to allow computers to form the control centre for a complete home entertainment system comprising video and audio components. The Pioneer PX-7 is the first MSX machine to be sold in the UK that can be used in this way. The addition of P-BASIC, an extension to standard MSX BASIC, allows software to control real video images and sound. These can be mixed with computer-generated graphics, thus opening the door to video adventures games, interactive video teaching programmes and more realistic simulations





CHRIS STEVENS

could go on to show some other section of film.

The LD-700 itself is a standard laser disc player and can be bought and used separately from the PX-7 computer. Each 12in disk can store up to 54,000 frames, and this means that it can show entire films, as well as offer single frame, slow and fast motion of a much higher quality than is possible with video. The remote controller supplied with the unit allows frame and chapter searching (a chapter being a division on the disc). Eacn disc carries two audio tracks, allowing it to store the soundtrack in two languages — although some discs use the second track to store a ready-made computer program.

Pioneer's hardware is well built and finished, although the computer ran very hot when operating in a stack with the laser disc player. Pioneer offers the PXTB-7 graphics tablet and a cartridge-based graphics pack called Video Art, allowing you to design programmes that use computer graphics superimposed on laser disc pictures. However, the standard of the software is disappointing — the package is slow, awkward to use and restricted in its abilities.

The PX-7 makes it very easy to create and run interactive video programs in the home, office or classroom. Writing appropriate software using the built-in extensions to BASIC is so easy it's almost trivial, and MSX BASIC is itself sophisticated and reasonably fast. Speed is not much of a problem however since the player takes about two seconds to search for a particular frame.

Rather than being a limited budget system, the PX-7 and LD-700 provide the potential for full interactive video programs. You could produce laser disc games, like those that have been so successful in amusement arcades, set up a vast pictorial database and so on. The system will also cope with CPE (computer program encoded) discs — those having a ready-made computer program stored on one of the two audio tracks.

However, the cost of mastering discs is likely to prohibit many from designing their own discs, and users will have to rely on commercial products from film companies or software houses. As is usual with new technology, it takes time for the software that will fully exploit the new hardware to become available. In the meantime, the Pioneer PX-7 is one of the most interesting MSX computers available, and coupled to a laser disc it represents the future direction of home entertainment and educational computing.

PIONEER PX-7

PRICE
PX-7 MSX computer: £299.99; LD-700 laser disc player: £499.90; PXI 32K RAM extension: £84.90; PXTB-7 graphics tablet: £89.90; joystick:£9.90 (all prices inc VAT)

DIMENSIONS
420×323×70mm

MEMORY
32K user RAM, expandable to 64K, 16K video RAM, 40K ROM
CPU
Z80 processor at 4MHz

SCREEN
40 columns × 24 lines, 16 colours, 256×192 high-resolution graphics with up to 32 sprites using 9929 display chip

SOUND
Stereo inputs and 3-voice sound using 8910 sound chip

INTERFACES
Composite video, TV, RGB, Centronics printer, 2 Joystick and 2 Cartridge ports, Cassette, Audio In, Audio Out, Headphones, System Control

LANGUAGES AVAILABLE
32K MSX BASIC and 8K P-BASIC

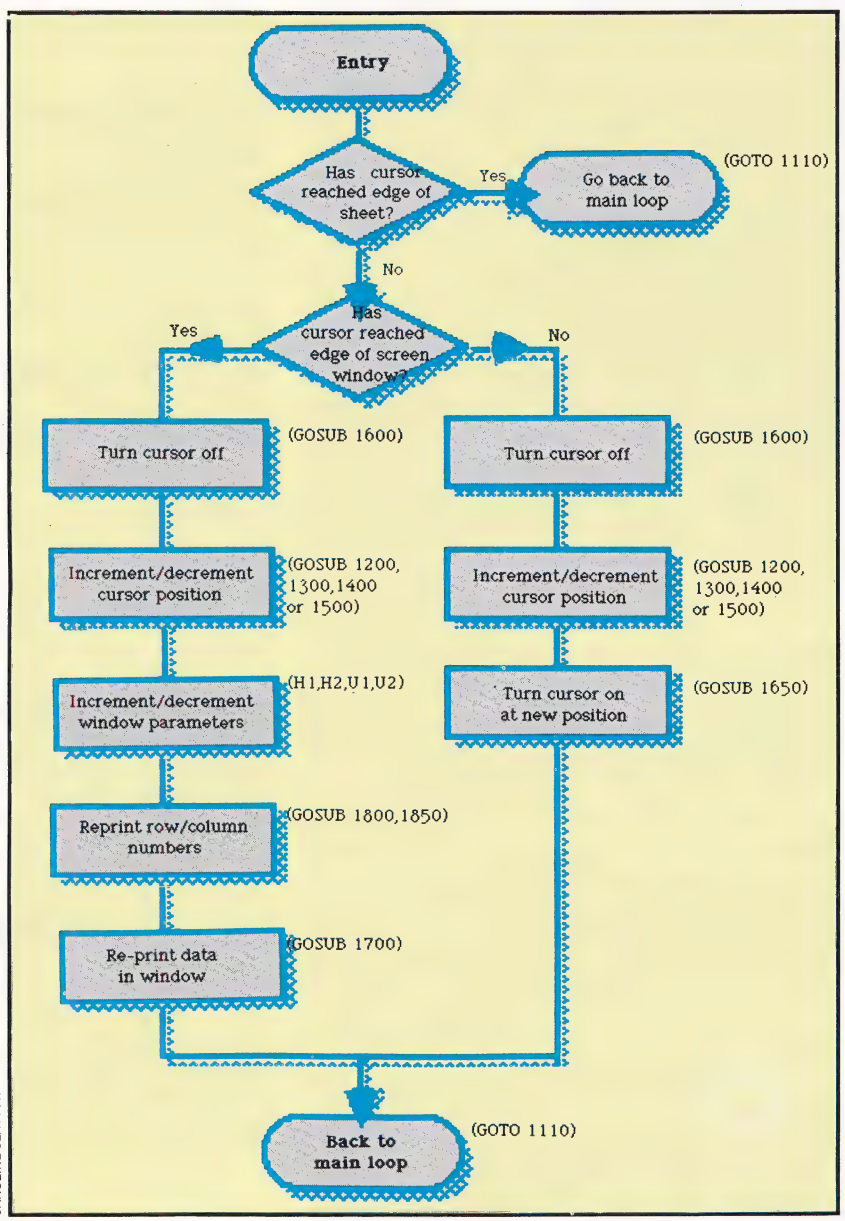
DOCUMENTATION
The PX-7 uses small booklets, such as you might expect to get with a hi-fi system. Some are very poorly translated from the Japanese and all the information is presented in a bland reference style. The manuals are complete however, and include large amounts of technical information

STRENGTHS
The PX-7 is the most developed MSX system available. Coupled with the laser disc it is a genuine advance in home entertainment technology. The units are well built, pleasantly designed and reasonably priced

WEAKNESSES
MSX has proved unpopular in the UK and has received minimum third-party support. The PX-7 also suffers because there is a shortage of software and laser discs to take advantage of its facilities

STARTING GRID

In the first instalment of this series, we looked at the design of spreadsheets in general, and gave the listings for the graphics routines for the Commodore 64. We turn our attention now to the programming of the screen displays for the Amstrad CPC 464/664, BBC Micro and Sinclair Spectrum.



CAROLINE CLAYTON

Close To The Edge
The flowchart above shows how our spreadsheet program controls the cursor on the 'sheet'. As the screen is only a window onto the spreadsheet, action must be taken to move on the next portion of the sheet when the cursor reaches the top, bottom, left or right edges of the visible screen area

Because the micros for which the spreadsheet has been designed have different methods of producing screen displays, we'll be giving the graphics routines for each independently. (The Commodore 64 version was given on page 1563.) The main functions of this part of the program are to print the spreadsheet grid on the screen, together with the row and column numbers, and to control the movement of the spreadsheet cursor across the grid.

Included in the cursor-handling routines are sections that move the cursor left, right, up and down. The screen can only display a portion of the spreadsheet at a time (it is useful to think of the screen as a window through which you can see only a part of the sheet), and so the cursor routines will also handle movement of the window across the spreadsheet.

Lines 1000 to 1080 form a subroutine that prints out the spreadsheet grid. The next section, beginning at line 1100, is the main program control loop, which essentially scans the keyboard for a keypress. Keys can be used to move the cursor around the sheet or to select a function, such as inputting a formula to a particular cell. The appropriate subroutine is thus called from this section.

CURSOR MOVEMENT

Most of the spreadsheet functions are the subject of later instalments and so attempting to select these at this stage will simply cause the program to crash. However, the subroutines included in this instalment — at lines 1200, 1300, 1400 and 1500 — will allow you to move the cursor around the screen, since they are the cursor-movement routines for RIGHT, LEFT, DOWN and UP, respectively. These four routines are similar, so let's just look at the first one to get an idea of how they all work.

On pressing the move-right cursor key, the program jumps to the subroutine at line 1200, and line 1210 checks to see whether the cursor has reached the edge of the sheet. It does this by testing the x co-ordinate to determine if it has reached its maximum value of 15. If this is the case, control is returned to the main program loop — otherwise, the routine continues. The next step is to see if the cursor is at the right-hand end of the current screen window. The variables H1 and H2 are used for the lower and upper horizontal limits of the window — for example, if H1=2 and H2=6, then columns 2 to 6 of the sheet are visible on the screen. If x has reached the value held in H2, the following chain of events is set in motion: the cursor is turned off, the value of x is incremented, the values of H1

and H2 are incremented, new row and column numbers are printed by the subroutine at line 1800 and the new cell data is printed onto the sheet. Finally, the cursor is turned back on.

DISPLAY OF THE CURSOR

The subroutines at lines 1600 and 1650 are of necessity different for each of the four computers, and they control the switching on and off of the spreadsheet cursor. In all versions, the cursor is shown by inverting the foreground and background colours in the appropriate cell, but the way in which this effect is achieved is peculiar to the display hardware and firmware of each machine.

On the Spectrum, foreground and background colours in each character cell are controlled by a byte in an area of memory called the 'attribute map'. The lowest three bits of the attribute byte control the INK colour and bits three to five control the PAPER colour. Thus, to reverse the colours, it is only necessary to locate the group of attribute bytes that correspond to the current spreadsheet cell and POKE in appropriate values.

On the Amstrad and BBC Micro versions, the process is a little trickier, because the value in the cell must be reprinted in the cell after turning the cursor on or off. The cell values are held in the array M(,); the correct value for the current cell must be found and converted to a string ready for printing. On the BBC Micro, the colours can be swapped using the COLOUR command. On the Amstrad, a control character, CHR$(24), is available and can be incorporated into a PRINT statement to exchange the current PEN and PAPER values.

Sinclair Spectrum:

```
  100 GO SUB 3000
  110 GO SUB 1000
  120 GO SUB 1700
  130 GO SUB 1100
  999 STOP
 1000 BORDER 1: PAPER 1: CLS : IN
K 7
 1005 PRINT "            C O L U M
 N S"
 1007 PRINT "ROW    1.    2.    3
.    4."
 1010 PRINT "-------+-----+-----+
-----+-----+"
 1020 FOR C=1 TO 6
 1030 PRINT CHR$ (C+64);".     |
    |     |     |     |"
 1040 PRINT "-------+-----+-----+
-----+-----+"
 1050 NEXT C
 1060 PRINT CHR$ (C+64);".     |
    |     |     |     |"
 1070 PRINT "-------+-----+-----+
-----+-----+"
 1080 RETURN
 1100 PRINT AT 0,0;"CELL:";CHR$ (
Y+64);STR$ (X)
 1110 LET A$=INKEY$: IF A$="" THE
N  GO TO 1110
 1120 IF A$="8" THEN  GO TO 1200:
 REM MOVE RIGHT
 1130 IF A$="5" THEN  GO TO 1300:
 REM MOVE LEFT
 1140 IF A$="6" THEN  GO TO 1400:
 REM MOVE DOWN
 1150 IF A$="7" THEN  GO TO 1500:
 REM MOVE UP
 1155 IF A$="C" THEN  GO SUB 2300
 : REM CALCULATE SHEET
 1160 IF A$="F" THEN  GO SUB 2000
 : REM INPUT FORMULA
 1165 IF A$="E" THEN  GO SUB 2100
 : REM ENTER NUMERIC DATA IN CELL
 1168 IF A$="H" THEN  GO TO 6000:
 REM PRINT HELP SCREEN
 1170 IF A$="S" THEN  GO SUB 5150
 : REM STORE CURRENT SHEET IN MEM
ORY
 1172 IF A$="G" THEN  GO SUB 5100
 : REM GET PREVIOUS SHEET
 1174 IF A$="Z" THEN  GO SUB 5000
 : REM CLEAR CURRENT SHEET
 1176 IF A$="R" THEN  GO SUB 5700
 : REM REPLICATE FORMULA
 1178 IF A$="T" THEN  GO SUB 5200
 : REM TAB TO NEW CELL
 1180 IF A$="D" THEN  GO SUB 7000
 : REM LOAD/SAVE DATA/FORMULAS
 1190 GO TO 1110
 1200 REM ***** MOVE RIGHT *****
 1210 IF X=15 THEN  GO TO 1100
 1220 IF X=H2 THEN  GO SUB 1600:
 LET X=X+1: LET H1=H1+1: LET H2=H
2+1: GO TO 1270
 1230 GO SUB 1600: LET X=X+1: GO
SUB 1650: GO TO 1100
 1270 GO SUB 1800: GO SUB 1700: G
O TO 1100
 1300 REM ***** MOVE LEFT *****
 1310 IF X=1 THEN  GO TO 1100
 1320 IF X=H1 THEN  GO SUB 1600:
 LET X=X-1: LET H1=H1-1: LET H2=H
2-1: GO TO 1370
 1330 GO SUB 1600: LET X=X-1: GO
SUB 1650: GO TO 1100
 1370 GO SUB 1800: GO SUB 1700: G
O TO 1100
 1400 REM ****** MOVE DOWN *****B
 1410 IF Y=15 THEN  GO TO 1100
 1420 IF Y=V2 THEN  GO SUB 1600:
 LET Y=Y+1: LET V1=V1+1: LET V2=V
2+1: GO TO 1470
 1430 GO SUB 1600: LET Y=Y+1: GO
SUB 1650: GO TO 1100
 1470 GO SUB 1850: GO SUB 1700: G
O TO 1100
 1500 REM ***** MOVE UP *******
 1510 IF Y=1 THEN  GO TO 1100
 1520 IF Y=V1 THEN  GO SUB 1600:
 LET Y=Y-1: LET V1=V1-1: LET V2=V
2-1: GO TO 1570
 1530 GO SUB 1600: LET Y=Y-1: GO
SUB 1650: GO TO 1100
 1570 GO SUB 1850: GO SUB 1700: G
O TO 1100
 1600 REM ** TURN CURSOR OFF ***
 1610 LET CU=22528+32*(V(Y+1-V1))
+H(X+1-H1)
 1615 POKE CU,15: POKE CU+1,15: P
OKE CU+2,15
 1620 POKE CU+3,15: POKE CU+4,15:
 RETURN
 1650 REM *** TURN CURSOR ON ***
 1660 LET CU=22528+32*(V(Y+1-V1))
+H(X+1-H1)
 1665 POKE CU,56: POKE CU+1,56: P
OKE CU+2,56
 1670 POKE CU+3,56: POKE CU+4,56
 1690 GO SUB 1900: RETURN
 1700 REM ** PRINT DATA IN SHEET
 1710 FOR I=0 TO 6
 1720 FOR J=0 TO 3
 1730 LET P$=STR$ (M(I+V1,J+H1))
 1740 PRINT AT V(I+1),H(J+1);"
  "
 1745 PRINT AT V(I+1),(H(J+1)+5-L
EN (P$));P$
 1750 NEXT J: NEXT I
 1760 GO SUB 1650: RETURN
 1800 REM *** PRINT COLUMN NOS **
 1810 FOR I=H1 TO H2
 1820 PRINT AT 1,7+6*(I-H1);I;".
"
 1830 NEXT I
 1840 RETURN
 1850 REM ** PRINT ROW NOS ****
 1870 FOR C=V1 TO V2
 1880 PRINT AT 2*(C-V1)+3,0;CHR$
(C+64);"."
 1890 NEXT C
 1895 RETURN
 1900 REM * FORMULA CURRENT CELL
 1920 LET D$=F$((Y-1)*15+X,1 TO )
 1930 PRINT AT 18,0;"FORMULA:
                    "
 1940 PRINT AT 18,0;"FORMULA: ";D
$
 1945 RETURN
```

Array Setup Routines:

```
 3000 REM ************************
 3001 REM * SETUP ARRAYS         *
 3002 REM ************************
 3010 DIM H(4): DIM V(7): DIM S(2
0): DIM S$(20): DIM E$(20): DIM
G$(20): DIM C(20)
 3020 FOR C=0 TO 3
 3030 LET H(C+1)=6*C+8: REM CALC
X-POS
 3040 NEXT C
 3050 FOR C=1 TO 7
 3060 LET V(C)=2*C+1: REM CALC Y-
POS
 3070 NEXT C
 3075 LET X=1: LET Y=1
 3080 LET H1=X: LET H2=X+3: LET V
1=1: LET V2=V1+6
 3090 REM ************************
 3091 REM * VALUE ARRAY          *
 3092 REM ************************
 3100 DIM M(15,15): DIM N(15,15)
 3110 FOR I=1 TO 15
 3120 FOR J=1 TO 15
 3130 LET M(I,J)=I*J
 3140 NEXT J: NEXT I
 3150 DIM F$(255,20): DIM G$(20):
 RETURN
```

BBC Micro:

```
   10REM **** BBC SPREADSHEET ****
   40 MODE 4
   50 *FX4,1
   60 LET CO$=CHR$(30):CL$=CHR$(8):CR$=C
HR$(9):CU$=CHR$(11):CD$=CHR$(10)
   70 REM VDU 23,1,0;0;0;0;
  100 GOSUB 3000:REM SETUP ARRAYS & SCRE
EN VARIABLE
  110 GOSUB 1000:REM PRINT SCREEN
  120 GOSUB 1700:REM PRINT DATA WINDOW O
N SCREEN
  130 GOTO 1100:REM MAIN KEYBOARD ROUTIN
E
  999 STOP
 1000 PRINT CHR$(12)
 1005 PRINT "               C O L U M N S
"
 1006 PRINT
 1007 PRINT "ROW     1.    2.    3.    4
.    5."
 1010 PRINT "         +-----+-----+-----+
-----+-----+"
 1020 FOR C= 1 TO 7
 1030 PRINT " "CHR$(C+64);".      |     |
     |     |     |     |"
 1040 PRINT " -------+-----+-----+-----+
-----+-----+"
 1050 NEXT C
 1060 PRINT " "CHR$(C+64);".      |     |
     |     |     |     |"
 1070 PRINT " -------+-----+-----+-----+
-----+-----+"
 1080 RETURN
 1100 P$=CHR$(Y+64)+STR$(X):PRINT CO$;CD
$"CELL:";P$;" "
 1110 LET A$=GET$
 1120 IF A$=CHR$(137) THEN 1200:REM MOVE
 CURSOR RIGHT
 1130 IF A$=CHR$(136) THEN 1300:REM MOVE
 CURSOR LEFT
 1140 IF A$=CHR$(138) THEN 1400:REM MOVE
 CURSOR DOWN
 1150 IF A$=CHR$(139) THEN 1500:REM MOVE
 CURSOR UP
 1152 IF A$="H"THEN GOSUB 6000:REM PRINT
 HELP SCREEN
 1154 IF A$="F" THEN GOSUB 2000:REM INPU
T FORMULA
 1156 IF A$="S"THEN GOSUB 5150:REM STORE
 CURRET SHEET
 1158 IF A$="G"THEN GOSUB 5100:REM GET P
REVIOUS SHEET
 1160 IF A$="C" THEN GOSUB 2300:REM CALC
ULATE SHEET
 1165 IF A$=CHR$(13) THEN RETURN
 1170 IF A$)="0" AND A$<="9" THEN GOSUB
2100:REM INPUT DATA ROUTINES
 1180 IF A$="Z" THEN GOSUB 5000:REM CLEA
R SHEET
 1185 IF A$="R" THEN GOSUB 5700:REM REPL
ICATE SHEET
 1187 IF A$="T" THEN GOSUB 5200:REM TAB
TO NEW CELL
 1189 IF (INKEY(-119)) THEN GOSUB 7000:R
EM LOAD SAVE ROUTINES
 1190 GOTO 1100:REM GO BACK TO START
 1200 REM ****** MOVE RIGHT ******
 1210 IF X=15 THEN 1100
 1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=H
1+1:H2=H2+1:GOTO 1270
 1230 GOSUB 1600:LET X=X+1:GOSUB 1650:GO
TO 1100
 1270 GOSUB 1800:GOSUB 1700:GOTO 1100
 1300 REM ******* MOVE LEFT *****
 1310 IF X=1 THEN 1100
 1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=H
1-1:H2=H2-1:GOTO 1370
 1330 GOSUB 1600:LET X=X-1:GOSUB 1650:GO
TO 1100
 1370 GOSUB 1800:GOSUB 1700:GOTO 1100
 1400 REM **** MOVE DOWN *****
 1410 IF Y=15 THEN 1100
 1420 IF Y=V2 THEN GOSUB 1600:LET Y=Y+1:
V1=V1+1:V2=V2+1:GOTO 1470
 1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GO
TO 1100
 1470 GOSUB 1850:GOSUB 1700:GOTO 1100
 1500 REM ***** MOVE UP *****
 1510 IF Y=1  THEN 1100
 1520 IF Y=V1 THEN GOSUB 1600:LET Y=Y-1:
V1=V1-1:V2=V2-1:GOTO 1570
 1530 GOSUB 1600:LET Y=Y-1:GOSUB 1650:GO
TO 1100
 1570 GOSUB 1850:GOSUB 1700:GOTO 1100
 1600 REM ****** TURN CURSOR OFF ****
 1610 P$=STR$(M(Y-V1+1,X-H1+1))
 1615 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1
615
 1620 COLOUR 1:COLOUR 128
 1625 PRINT TAB(H(X-H1+1)-1,V(Y-V1+1)-1)
;P$
 1630 COLOUR 1:COLOUR 128
 1640 RETURN
 1650 REM ****** TURN CURSOR ON *****
 1660 P$=STR$(M(Y-V1+1,X-H1+1))
 1665 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1
665
 1670 COLOUR 2:COLOUR 129
 1675 PRINT TAB(H(X-H1+1)-1,V(Y-V1+1)-1)
;P$
 1680 COLOUR 1:COLOUR 128
 1690 GOSUB 1900:RETURN
 1700 REM ***** PRINT DATA WINDOW FROM S
HEET ON SCREEN ****
 1710 FOR I=0 TO 7
 1720 FOR J=0 TO 4
 1730 P$=STR$(M(I+V1,J+H1))
 1735 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 1
735
 1740 PRINT TAB(H(J+1)-1,V(I+1)-1);"
 "
 1745 PRINT TAB(H(J+1)-1,V(I+1)-1);P$
 1750 NEXT J,I
 1760 GOSUB 1650:RETURN
 1800 REM ***** PRINT COLUMN NO. ****
 1810  FOR I=H1 TO H2:PRINT TAB(8+6*(I-H
1),3);I;"."
 1820 NEXT I:RETURN
 1850 REM ****** PRINT ROW LABELS *****
 1860 FOR C=V1 TO V2
 1870 PRINT TAB(1,V(C-V1+1)-1);CHR$(C+64
);"."
 1880 PRINT:NEXT C:RETURN
 1900 REM **** FORMULA OF CURRENT CELL *
**
 1910 LET D$=F$((Y-1)*15+X)
 1920 PRINT TAB(0,22);"
                      "
 1930 PRINT TAB(0,22);"FORMULA: ";D$
 1940 RETURN
```

Array Setup Routines:

```
 3000 REM ***** SETUP ARRAYS ******
 3010 DIM H(5),V(8),ST(20),ST$(20),E$(20
),G$(20),C(20)
 3020 FOR C=0 TO 4
 3030 LET H(C+1)=6*C+10
 3040 NEXT C
 3050 FOR C=1 TO 8
 3060 LET V(C)=2*C+4:REM CALC YPOS
 3070 NEXT C
 3075 X=1:Y=1:REM INITIAL CURSOR POSITIO
 3080 H1=X:H2=X+4:V1=Y:V2=Y+7
 3090 REM ***** DIM SHEET ARRAYS ****
 3100 DIM M(15,15):DIM N(15,15)
 3110 FOR I=1 TO 15:FOR J=1 TO 15
 3120 LET M(I,J)=I*J+I
 3130 NEXT J,I
 3140 DIM F$(225)
 3150 RETURN
```

Amstrad CPC 464/664:

```
100 GOSUB 3000:REM SETUP ARRAYS & VARIA
BLES
110 GOSUB 1000:REM PRINT SCREEN
120 GOSUB 1700:REM PRINT DATA ON SCREEN
130 GOTO 1100
999 STOP
1000 REM PRINT SCREEN DISPLAY
1005 CLS:PRINT "               C O L U M
N S"
1006 PRINT
1007 PRINT "ROW      1.     2.     3.     4.
    5."
1010 PRINT "         +------+------+------+-
-----+------+"
1020 FOR C=1 TO 7
1030 PRINT " "CHR$(C+64);".      |      |
    |      |      |      |"
1040 PRINT " --------+------+------+------+-
-----+------+"
1050 NEXT C
1060 PRINT " "CHR$(C+64);".      |      |
    |      |      |      |"
1070 PRINT " --------+------+------+------+-
-----+------+"
1080 RETURN
1100 LOCATE 1,1:P$=CHR$(Y+64)+MID$(STR$(
X),2,2):PRINT "CELL ";P$;" "
1110 A$=INKEY$:IF A$="" THEN 1110
1120 IF A$=CHR$(243) THEN 1200:REM MOVE
RIGHT
1130 IF A$=CHR$(242) THEN 1300:REM MOVE
LEFT
1140 IF A$=CHR$(241) THEN 1400:REM MOVE
DOWN
1150 IF A$=CHR$(240) THEN 1500:REM MOVE
UP
1155 IF A$="F" THEN GOSUB 2000:REM INPUT
 FORMULA
1160 IF A$="C" THEN GOSUB 2300:REM CALCU
LATE SHEET
1165 IF A$=CHR$(13) THEN RETURN
1170 IF A$>="0" AND A$<"9" THEN GOSUB 21
00:REM ENTER NUMERIC DATA
1180 IF A$="B" THEN GOSUB 5000:REM CLEAR
 SHEET
1185 IF A$="V" THEN GOSUB 5100:REM GET P
REVIPOUS SHEET
1187 IF A$="G" THEN 5200:REM MOVE CURSOR
 TO NEW CELL
1188 IF A$="R" THEN GOSUB 5700
1190 GOTO 1100
1200 REM **** MOVE RIGHT ****
1210 IF X=15 THEN 1100
1220 IF X=H2 THEN GOSUB 1600:X=X+1:H1=H1
+1:H2=H2+1:GOTO 1270
1230 GOSUB 1600:LET X=X+1:GOSUB 1650:GOT
O 1100
1270 GOSUB 1800:GOSUB 1700:GOTO 1100
1300 REM **** MOVE LEFT ***
1310 IF X=1 THEN 1100
1320 IF X=H1 THEN GOSUB 1600:X=X-1:H1=H1
-1:H2=H2-1:GOTO 1370
1330 GOSUB 1600:LET X=X-1:GOSUB 1650:GOT
O 1100
1370 GOSUB 1800:GOSUB 1700:GOTO 1100
1400 REM **** MOVE DOWN ****
1410 IF Y=15 THEN 1100
1420 IF Y=V2 THEN GOSUB 1600:Y=Y+1:V1=V1
+1:V2=V2+1:GOTO 1470
1430 GOSUB 1600:LET Y=Y+1:GOSUB 1650:GOT
O 1100
1470 GOSUB 1850:GOSUB 1700:GOTO 1100
1500 REM *** MOVE DOWN ****
1510 IF Y=1 THEN 1100
1520 IF Y=V1 THEN GOSUB 1600:Y=Y-1:V1=V1
-1:V2=V2-1:GOTO 1570
1530 GOSUB 1600:LET Y=Y-1:GOSUB 1650:GOT
O 1100
1570 GOSUB 1850:GOSUB 1700:GOTO 1100
1600 REM *** TURN CURSOR OFF ****
1610 LET P$=MID$(STR$(M(Y,X)),2)
1615 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 16
15
1620 LOCATE H(X-H1+1)-1,V(Y-V1+1):PRINT
P$
1630 RETURN
1650 REM *** TURN CURSOR ON ****
1660 LET P$=MID$(STR$(M(Y,X)),2)
1665 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 16
65
1670 LOCATE H(X-H1+1)-1,V(Y-V1+1):PRINT
CHR$(24);P$;CHR$(24);
1680 GOSUB 1900:RETURN
1700 REM **** PRINT DATA IN SHEET ****
1710 FOR I=0 TO 7
1720 LOCATE 10,V(I+1)
1730 FOR J=0 TO 4
1735 LET P$=MID$(STR$(M(I+V1,J+H1)),2)
1740 IF LEN(P$)<5 THEN P$=" "+P$:GOTO 17
40
1745 LOCATE H(J+1)-1,V(I+1):PRINT P$;
1750 NEXT J,I
1760 GOSUB 1650:REM TURN CURSOR ON
1770 RETURN
1800 REM ***** PRINT COLUMN NOS ****
1810 LOCATE 1,3:PRINT "ROW      ";;
1820 LOCATE 7,3:FOR I=H1 TO H2:PRINT TAB
(H(I-H1+1)-3)I;CHR$(8);".";
1830 NEXT I
1840 RETURN
1850 REM ***** PRINT ROW LETTERS *****
1860 LOCATE 1,4
1870 FOR I=V1 TO V2
1875 PRINT
1880 PRINT " ";CHR$(I+64);"."
1890 NEXT I
1895 RETURN
1900 REM ***** FORMULA OF CURRENT CELL *
**
1920 LET D$=F$((Y-1)*15+X)
1930 LOCATE 1,22
1940 PRINT "FORMULA:
          "
1950 PRINT  '

FORMULA:";D$
1960 LOCATE 1,1
1970 RETURN
```

Array Setup Routines:

```
3000 REM ******************************
3001 REM * SETUP ARRAYS & VARAIBLES   *
3002 REM ******************************
3010 DIM H(5),V(8),ST(20),ST$(20),E$(20)
,G$(20),C(20)
3020 FOR C= 0 TO 4
3030 H(C+1)=6*C+11:REM CALC XPOS
3040 NEXT C
3050 FOR C=1 TO 8
3060 LET V(C)=2*C+3:REM CALC YPOS
3070 NEXT C
3075 LET X=1:Y=1
3080 LET H1=X:H2=X+4:V1=Y:V2=Y+7
3090 REM ******************************
3091 REM * VALUE ARRAY                 *
3092 REM ******************************
3100 DIM M(15,15):DIM N(15,15)
3110 FOR I=1 TO 15
3120 FOR J=1 TO 15
3130 LET M(I,J)=I*J
3140 NEXT J,I
3150 REM * FORMULA ARRAY               *
3152  REM ******************************
3160 DIM F$(255)
3170 LET F$(1)="A1+B1+C1"
3180 LET F$(31)="C1+C2+C3"
3190 LET F$(16)="B1+B2+B3"
3200 RETURN
```

OBJECT PROGRAM

In the previous instalment, we examined the tree structures required for object manipulation (see page 1575). We give here the listings that enable these structures to be entered into our program, and suggest some alterations to liven up the Dog and Bucket.

Our object manipulation tree can be programmed as shown by adding the various lines to the core listings printed on pages 1507 and 1508. The key lines here are lines 210 and 220, 2430 and 2440, and 5030 to 5090. Let's examine each of these key pieces of code in turn.

First, lines 210 to 220 set up the arrays needed to store the data for our tree. Remember that unlike some of the earlier trees we examined in this series, this tree tests many different conditions, regardless of the level of descent or the node number. We therefore need to store for each node a record of the conditional value it is testing and the nodes it will lead to, depending on whether the condition is true or false. The c array holds the different conditional values, and each node tests an element of that array. The number of the element to be tested is read into the array k(number of trees, max number of choice nodes).

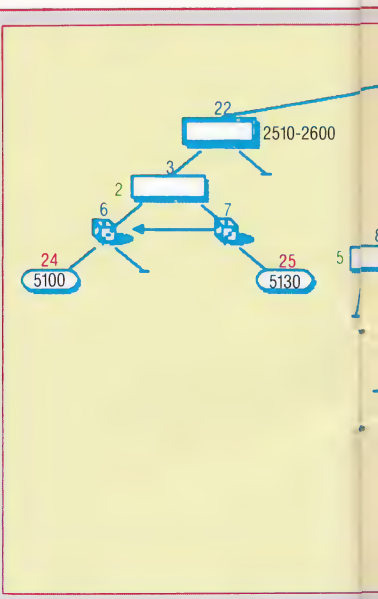
Lines 2430 and 2440 initialise the c array. These lines constitute a subroutine that must be called for each character, since the value of the conditions will obviously vary for each individual case.

We traverse the tree in lines 5030 to 5060. Line 5040 checks the current node number, and if it is a terminal node (that is, if it is numbered higher than 21) then it jumps out of the tree to select the routine at lines 5070 to 5090. Line 5050 checks to see if the node is testing condition 12, which indicates a random node, and if so calls the random number routine to assign a value (either one or two) to the condition. Line 5060 then performs the most important part of the operation, selecting the new node number from the t array and then jumping back to line 5040.

Running the complete program will show you the character handler in action. Enter Y in answer to the prompt Default values? and see what happens. The character editor at line 2350 is not fully compatible with the character handler as it stands. We shall examine its application in more detail in the next instalment. You can change location by pressing 1, 2 or 3.

At this stage, you may find the action somewhat repetitive, but this will soon change when we add the final two routines — the 'interaction' routine and the 'plot' routine — in the remaining instalments.

The following lines must be added to the initialisation module. The functions are useful for manipulating the data held in the two main string arrays. Line 190 sets up an array, t, which will be used to store the data for our three main tree structures — the 'object' tree (in this instalment), the 'interaction' tree and the 'plot' tree (discussed in the following instalment). The array k holds the different conditions that must be tested at the various nodes, and the array c holds the conditional values (which are initialised in lines 2430 and 2440)



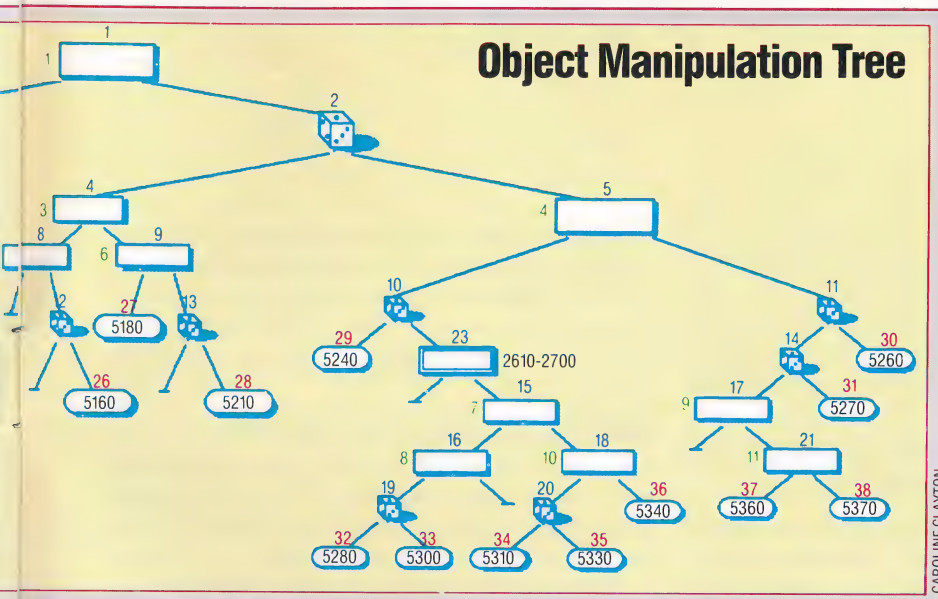

```
130 DEF FNb(y,z)=VAL(b$(y,z))
140 DEF FNc(y,z)=VAL(c$(y,z))
150 DEF FNm$(c$,d)=STR$((VAL(c$))-d)
160 DEF FNi$=b$(VAL(c$(c,3)),1)
180 REM setup trees
190 DIM t(3,25,2),k(3,30),c(25)
200 REM object tree
210 FOR n=1 TO 21: REM 21 choice nodes
220 READ k(1,n),t(1,n,2),t(1,n,1): NEXT n
```

We need to adjust our main program loop to take into account the first part of our character handler. Lines 550 to 800 take each character in turn, check to see if the 'handle' flag is greater than zero (line 560), and, if it is, decreases it by one and proceeds to the next character. If the value is zero, then the character handle factor (c$(n,10)) is reset by reading through the data in lines 6030 and 6040. Line 580 checks to see if the default handle value is zero, in which case the character is not to be processed by the handler at all. Line 590 initialises the conditions for the tree by calling the subroutine at 2430, and then calls the handler at line 5000

```
500 REM
510 REM test program loop
520 REM
530 GOSUB 2100: GOSUB 2150: GOSUB 2240:
PRINT: PRINT
540 REM character handler
550 FOR c=1 TO 6
560 IF FNc(c,10)>0 THEN c$(c,10)=FNm$(c$
(c,10),1): GOTO 800
570 RESTORE: FOR n=1 TO c*10+c-1: READ c
$(c,10): NEXT n: REM reset default handl
e value
580 IF FNc(c,10)=0 THEN GOTO 800
590 GOSUB 2430: GOSUB 5000: REM call obj
ect tree
800 NEXT c
810 GOSUB 4260: IF i$="" GOTO 550
820 GOSUB 2040: GOTO 530
```

Lines 2430 and 2440 assign to the c array the various conditional values that will be tested during traversal of the tree. The subroutines at lines 2520, 2620 and 2720 are called at various points by the character handler

```
2400 REM
2410 REM conditions
2420 REM
2430 h=FNc(c,8): i=FNc(c,3): j=FNc(c,6):
 c(1)=ABS(i)0): c(2)=ABS((FNb(j,2)=FNc(c
,2)) AND (q=1)): c(3)=ABS(b$(i,3)="y"):
c(4)=ABS(FNc(c,3)=FNc(c,6)): c(5)=ABS(b$
(i,4)="y")
2440 c(6)=ABS(i=3): c(7)=ABS(FNc(c,5))5)
: c(8)=ABS(FNc(c,5))2): c(9)=ABS(VAL(c$(
c,9))=1): c(10)=ABS(FNc(x,3)=0): c(11)=A
BS(FNc(h,2)=FNc(c,2)): c(12)=255
2500 RETURN
2510 REM
2520 REM check location for objects
2530 REM
2540 f=0: REM set 'found flag' to zero
2550 FOR b=1 TO 12
2560 IF FNb(b,2)=FNc(c,2) THEN f=1: b=12
2570 NEXT b
2580 IF f=1 THEN n=3: GOTO 2600
2590 n=39
2600 RETURN
2610 REM
2620 REM check for presence of owner of
object: if present, set x to character n
umber: jump back into tree
2630 REM
2640 f=0:x=0
2650 FOR m=1 TO 6
```



CAROLINE CLAYTON

Branch Lines

We give here the object manipulation tree, with choice nodes numbered in blue and terminal nodes in red. Nodes 22 and 23 jump to subroutines that determine the new node number. Each choice node also shows (in green) the number of the c array element that holds the value of the condition to be tested (see lines 2430 and 2440); random nodes have all been assigned the element 12. The terminal nodes are labelled with the line number of the subroutine to which control is passed by the main program after traversing the tree. Spectrum flavours for these listings will be given in the next instalment

```
2660 IF (FNc(m,2)=FNc(c,2)) AND (FNc(m,6
)=FNc(c,3)) THEN f=1: x=m: GOSUB 2430: m
=6
2670 NEXT m
2680 IF f=1 THEN n=15: GOTO 2700
2690 n=39
2700 RETURN
2710 REM
2720 REM select object at random from ch
aracter's location
2730 REM
2740 b=0
2750 FOR s=1 TO 12
2760 IF FNb(s,2)<>FNc(c,2) THEN GOTO 278
0
2770 GOSUB 4180: IF q=1 THEN b=s: s=12
2780 NEXT s
2790 IF b=0 THEN GOTO 2750
2800 RETURN
```

We add here a random number routine for the various random choice nodes, as well as a short routine to zero the codes in c$(n,8) and c$(n,9)

```
4150 REM
4160 REM random number routine
4170 REM
4180 q=INT(RND(1)*2)+1: RETURN
4190 REM
4200 REM zero character codes
4210 REM
4220 c$(c,8)="0": c$(c,9)="0": RETURN
4230 REM
4240 REM test to see if key pressed
4250 REM
4260 i$=INKEY$: RETURN
```

These lines traverse the tree (5030 to 5060) and select the various subroutines as determined by the terminal nodes (line 5090). Note line 5080, which selects two subroutines that return new node values and then jumps back into the tree traversal process

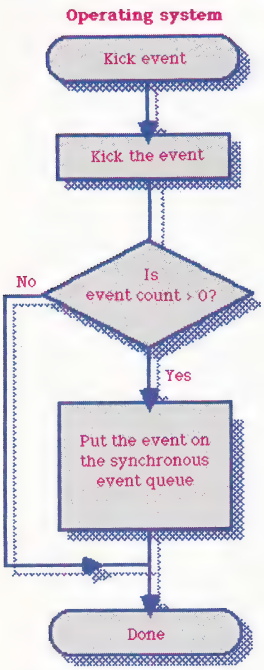
```
5000 REM object tree routines
5010 p=0: REM zero print flag
5020 IF FNc(c,2)=r THEN p=1
5030 n=1: REM start at node 1
5040 IF n>21 GOTO 5070
5050 k=c(k(1,n))+1: IF k(1,n)=12 THEN GO
SUB 4180: k=q
5060 n=t(1,n,k): GOTO 5040
5070 IF n>=24 GOTO 5090
5080 ON (n-21) GOSUB 2540,2640: GOTO 504
0
5090 ON (n-23) GOTO 5100,5130,5160,5180,
5210,5240,5260,5270,5280,5300,5310,5330,
5340,5360,5370,5430
5100 GOSUB 2740: c$(c,3)=STR$(b)
5110 IF p=1 THEN PRINT c$(c,1);" picks u
p";b$(b,1): PRINT
5120 b$(b,2)="0": c$(c,9)="4": RETURN
5130 c$(c,3)=c$(c,6)
5140 IF p=1 THEN PRINT c$(c,1);" picks u
p";FNi$: PRINT
5150 b$(VAL(c$(c,3)),2)="0": c$(c,9)="4"
: RETURN
5160 IF p=1 THEN PRINT c$(c,1);" takes a
 sip from";FNi$: PRINT
5170 c$(c,4)=FNm$(c$(c,4),-1): RETURN
5180 GOSUB 4180: IF (p=1) AND (q=1) THEN
 PRINT c$(c,1);" is eating the sandwich.
": PRINT
5190 c$(c,4)=FNm$(c$(c,4),-2): c$(c,9)="
6": GOSUB 4180: IF q=1 THEN GOSUB 4220
5200 RETURN
5210 IF p=1 THEN PRINT c$(c,1);" takes a
 tentative bite of the pasty, groans, an
d drops it on the floor.": PRINT
5220 g=c: REM set pasty eaten flag
5230 c$(c,3)="0": c$(c,4)=FNm$(c$(c,4),1
0): b$(3,2)=c$(c,2): RETURN
5240 IF p=1 THEN PRINT c$(c,1);" puts do
wn";FNi$: PRINT
5250 b$(VAL(c$(c,3)),2)=c$(c,2): c$(c,3)
="0": RETURN
5260 c$(c,5)=FNm$(c$(c,5),-1): RETURN
5270 GOSUB 5240: RETURN
5280 IF p=1 THEN PRINT c$(c,1);" throws"
;b$(VAL(c$(c,3)),1);" at ";c$(x,1): PRIN
T
5290 c$(x,4)=FNm$(c$(x,4),1): b$(VAL(c$(
c,3)),2)=c$(c,2): c$(x,8)=STR$(c): c$(x,
9)="5": c$(c,3)="0": RETURN
5300 GOSUB 4220: RETURN
5310 IF p=1 THEN PRINT "I think I've got
 your drink, says ";c$(c,1);" to ";c$(x,
1): PRINT
5320 c$(c,8)=STR$(x): c$(c,9)="2": RETUR
N
5330 c$(c,4)=FNm$(c$(c,4),2): RETURN
5340 IF p=1 THEN PRINT c$(c,1);" gives";
FNi$;" to ";c$(x,1): PRINT
5350 c$(x,3)=c$(c,3): c$(c,3)="0": c$(x,
8)=STR$(c): c$(x,9)="1": RETURN
5360 GOSUB 4220: RETURN
5370 IF p=0 GOTO 5420
5380 IF p=1 THEN PRINT c$(c,1);" is drun
kenly thanking ";c$(VAL(c$(c,8)),1);" fo
r returning ";
5390 IF p=1 AND c$(c,7)="f" THEN PRINT "
her ";: GOTO 5410
5400 PRINT "his ";
5410 PRINT "drink": PRINT
5420 GOSUB 4220: RETURN
5430 RETURN
```

Line 6230 holds the data for the object manipulation tree. The values are fetched in groups of three by line 220, which assigns for each node in turn: the subscript of the c array element that holds the condition to be tested; the number of the node to be branched to if the condition is true; and the node to be jumped to if the condition is false

```
6200 REM
6210 REM object tree data
6220 REM
6230 DATA 1,2,22,12,5,4,2,7,6,3,9,8,4,11
,10,12,39,24,12,6,25,5,12,39,6,13,27,12,
23,29,12,30,14,12,26,39,12,28,39,12,31,1
7,7,18,16,8,39,19,9,21,39,10,36,20,12,33
,32,12,35,34,11,38,37
```

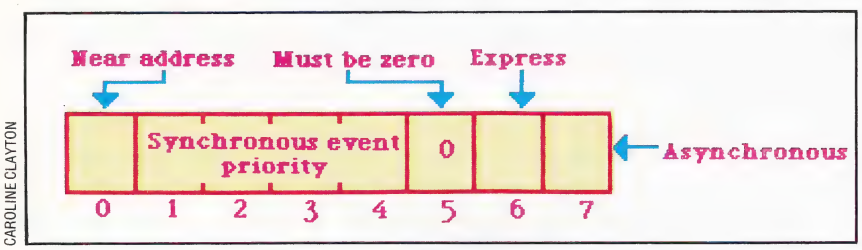
AFTER THE EVENT

Synchronicity
The two flow diagrams below show the sequence of operations performed by the operating system (top) and the main program (bottom) for the processing of synchronous events

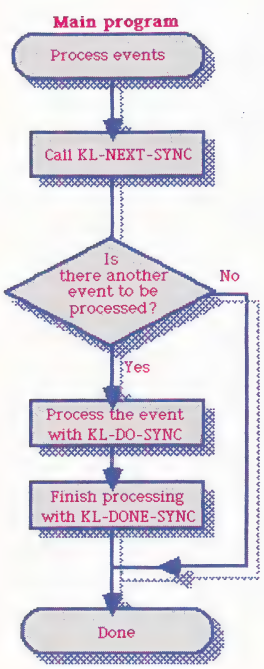


In the previous instalment we looked at the Amstrad operating system's use of interrupts — the method of system timing controlled by the hardware — and introduced events, the software equivalent of interrupts. Here, we conclude our discussion of the usefulness of events to the Amstrad machine code programmer.

We have already described how the Amstrad OS accesses software events via an 'event block', which consists of seven contiguous bytes located anywhere within the central 32 Kbyte block of RAM. Event blocks are set up by the user by calling the routine KL_INIT_EVENT at $BCEF, having previously reserved the seven bytes required. The routine is called with the HL register pair containing the address of the block, B containing the event class (in bit-significant form, as shown below), C containing the ROM select (0 if in RAM); and DE holding the address of the event routine to be called. The listing provided gives an example of this operation.



CAROLINE CLAYTON



The event class, as passed in the B register, is shown in the diagram. Once an event has been initialised, exactly when the routine is called by the operating system depends on the type (synchronous/asynchronous) and priority (normal/express) of the event. Generally, the routine to be called may be anywhere in RAM or in any ROM and the address of the user field in the event block is passed to the event routine for its own use.

The way that the operating system deals with synchronous and asynchronous events differs considerably, so they are best described separately — although, of course, both types of event may be used for any given application.

The 'kicks' — increments of the event count — may come from one of four distinct sources: the fast ticker interrupt, the ticker interrupt, the frame flyback interrupt or the jumpblock entry KL_EVENT. The three timer interrupts each have an associated list of events that need to be kicked when the interrupt occurs. The routines to set up the lists are called via jumpblock entries. It is possible to either initialise a new event block and add it to a list, or to add an existing block to any of the lists. Event blocks may be set and initialised separately by using KL_INIT_EVENT. The routine KL_EVENT is general purpose and may be used to kick any event.

ASYNCHRONOUS EVENTS

Associated with *normal* asynchronous events is an interrupt event pending queue. This queue is used to hold all the asynchronous events that have been kicked during the external interrupt.

As an example consider a normal asynchronous event block that has been arranged to be kicked by the ticker interrupt. When the ticker interrupt occurs, the operating system looks for any events that need to be kicked. In this case, our event will be kicked. If, after kicking, the event count is found to be greater than zero, then the event will not be dealt with immediately but will be placed on the interrupt event pending queue. After the operating system has performed all the tasks associated with the ticker interrupt, each routine on the event pending queue will then be called in turn. Because the ticker interrupt is re-enabled before the event routines are called, any further kicks will be noted. Therefore, the routine may take as long as it needs without missing any further kicks. After an event routine has been called its count is decremented.

If an *express* asynchronous event is kicked and has a count greater than zero, then it is not put on the event pending queue but its event routine is called immediately while the interrupts are disabled. However, if the event routine is too long then any further external interrupts will be missed — therefore, the routine should be as short as possible. This type of event is generally not used.

SYNCHRONOUS EVENTS

The main program decides when synchronous events should be processed. Because of this, the operating system simply kicks the event and, if greater than zero, puts it onto the synchronous event pending queue according to the priority assigned to it in the event block. Express synchronous events are merely arranged to have priority over the normal type.

The operating system provides several jumpblock entries to enable a main program to process synchronous events, to optionally clear the queue and to prevent particular events from occurring. Additionally, all normal synchronous events may be disabled if need be.

When the main program decides to process synchronous events it performs the tasks outlined

in the flow diagrams. Firstly, the KL_NEXT_SYNC jumpblock entry is called to obtain the next outstanding event from the queue. The event is then processed by calling KL_DO_SYNC; this routine looks up the address of the event routine from the event block and calls it. The KL_DONE_SYNC entry is then called to signal to the operating system that the processing for that event has finished. At this stage the event count is decremented and if the count remains greater than zero the event block is placed back on the synchronous event queue.

DISABLING EVENTS

At some point it will be necessary to disable or prevent particular events from occurring. The operating system allows for this with a variety of jumpblock entries. For asynchronous events, the entry KL_DISARM_EVENT sets the event count for the given event block to a negative value, thereby preventing any further kicks from causing the event routine to be called.

Three entries allow synchronous events to be disabled. The first, KL_SYNC_RESET, clears all outstanding events from the synchronous event-pending queue but does not alter the event count. This effectively disarms the event, since the event count cannot be decremented unless the event is on the queue. Another entry — KL_DEL_SYNCHRONOUS — disarms and removes any event that happens to be in the event queue. The final entry, KL_EVENT_DISABLE, stops any normal synchronous events form being put in the queue, but still allows the events to be kicked.

Events may appear complex at first, but they protect the programmer from the complications normally associated with interrupts, although care must still be taken when using express asynchronous events to avoid data clashing with the main program.

Background Beeper

The listing given here is an example of how to use events, incorporating the techniques described in the main text. A ticker event is set up and used as a timer, which causes one event to count seconds. This in turn sets up an event that makes the computer beep once every second. The program should be called at its Assembly address to initialise the events.

Notice that the beeping continues in the background when a BASIC program is running (although strange effects may occur if the BASIC program itself uses the sound generator)

```
; This program demonstrates the use of several
; types of event to generate a simple clock that
; sounds a beeper every second.
;
INIT.EVENT:     EQU  £BCEF ; KL_INIT_EVENT
ADD.TICK:       EQU  £BCE9 ; KL_ADD_TICKER
KICK:           EQU  £BCF2 ; KL_EVENT
TXT.OUT         EQU  £BB5A ; TXT_OUTPUT
;
BLEEP:           EQU  £07
;
; First initialise the event blocks
;
                ORG  £8000
;
                LD   HL, TICK  ; start with ticker
                LD   B, £81    ; async, express
                LD   C, 0      ; no rom select
                LD   DE, FRAC  ; routine address
;
                CALL INIT.EVENT
;
; BC is preserved
;
                LD   HL, SEC   ; seconds counter
                LD   DE, SECND ; routine address
                CALL INIT.EVENT
;
                LD   HL, BP    ; bleeper routine
                LD   DE, BEEP  ; routine address
                CALL INIT.EVENT
;
; now log on the Ticker Routine
;
                LD   HL, FRBLK ; tick block address
                LD   DE, 1     ; count
                LD   BC, 1     ; recharge
                CALL ADD.TICK  ; log it in
;
                RET
;
;Event Processing Routines
;
FRAC:
; maintains a 1/50th of a second count
;
                LD   HL, SEC50 ; point to count
                LD   B, 50     ; one second?
                CALL TEST
                RET  NZ        ; no, then finish
                LD   HL, SEC   ; kick next second
                CALL KICK
                RET
;
SECND:
; This is called once a second - kicks a beep
                LD  HL, BP     ; kick beep event
                CALL KICK
                RET
;
BEEP:
; does a beep
                LD   A, BLEEP  ; send a bleep char
                CALL TXT.OUT
                RET
;
TEST:
; tests a counter
; B contains the required value on entry
; HL points to the count store location
;
                INC  (HL)      ; update it
                LD   A, (HL)   ; read the count
                CP   B         ; finished
;
                RET  NZ        ; no, return
                XOR  A         ; set zero true
                LD   (HL), A   ; reset the count
                RET            ; and go back
;
; now the event and tick blocks
;
FRBLK:          DEFS 6         ; tick block space
;
TICK:           DEFS 7         ; ticker event block
;
SEC:            DEFS 7         ; 1 sec event block
;
BP:             DEFS 7         ; beep event block
;
SEC50:          DEFB 0         ; 1/50 sec counter
```

SERIAL INPUT/OUTPUT

In an eight-bit computer, most transmissions are performed via parallel data lines, though often when a computer is communicating with peripherals or other attached devices, the machine will use serial transmission when the bits need to be sent sequentially. Therefore, in order to convert the signal from parallel to serial format, *serial input/output* techniques have to be employed.

When a computer receives a serial data stream, the information is read into a shift register. This can be of any length but is usually eight bits long (or a multiple of eight). The amount of data arriving is measured against an external clock, and when the shift register is full, the computer activates a read enable line and the information will then be transferred onto the machine's parallel data bus. The shift register will then be ready to receive the next set of data from the serial line.

It is often necessary to re-transmit the data through a serial output line. In this case, the process is reversed. The eight-bit parallel data is fed into another shift register, which is again synchronised by a clock. Once all the bits are transmitted down the serial interface, the process may be repeated. These serial conversion circuits are kept on a single chip known as a UART (universal asynchronous receiver transmitter).

SERVER

This is a node within a local area network (LAN; see page 969) that handles the management of a peripheral or supervises the entire network. *Server* stations are particularly useful when large pieces of equipment aren't needed by each computer all the time.

There are several basic types of server. A 'file server' will control disk drives (usually hard disks) through which each computer within the network can gain access to the information held on the disks. Similarly, a 'printer server' will be connected to a printer, enabling resources, which might otherwise be devoted to providing each terminal with its own printer, to be devoted instead to a single high-speed, high-quality device. A 'communications server' is used to link the local area network to other LANs (via a modem), public databases or other mainframe computers.

SERVO MOTOR

A type of motor widely used to perform mechanical work under computer control, *servo motors* are particularly useful in such applications because they move in response to digital signals received from a computer, in the form of pulses. Typically, the servo motor has three lines connected to it: a power supply that varies depending on the size of the motor, a control line through which the pulses are transmitted and a common return line for both the power and control lines.

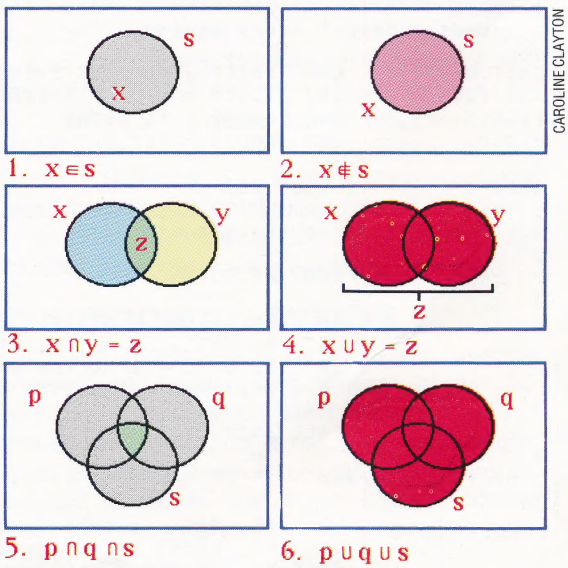
The angle of movement of the servo motor depends on the length of the pulse transmitted from the computer, which can vary, but is generally between one and two milliseconds. These values correspond to the minimum and maximum angles of rotation permitted by the servo motor, and the longer the pulse, the greater the angle of movement from the minimum position. It should be noted that servo motors cannot rotate through 360°. In order to maintain the motor at a given position, it is necessary to send a 'refresh' signal via the control line at regular intervals, which is generally every 20 milliseconds a transmission frequency of 50Hz.

SET

Any group of elements which have been collected together is called a *set*. Elements in a set are usually written as $x \in S$, where x is a member of set S. Conversely, if x is not an element in set S, we can write $x \notin S$. Sets having the same members and the same number of members are called 'identical sets'.

Sometimes sets will not be identical but they may have some members in common. This group of common members is known as the intersection between the sets, and is written $x \cap y = z$, where x and y are sets and z is the intersection. Alternatively, when sets are joined together to form a common set, it is known as the union of the sets and written as $x \cup y = z$, where z is the union. Finally, the elements in one set y may all be members of a second set x, although the sets may not be identical. In this case, we can say that set y is a subset of set x, and can be written as $y \subset x$.

Sets are particularly useful in describing logical operations. For example, an intersection between two sets is equivalent to a Boolean AND operation, while a union is equivalent to OR.



CAROLINE CLAYTON

Set Points

Sets are represented pictorially in Venn diagrams. The oblong border of each diagram is regarded as the boundary of the Universal set (the set from which the members of any set can be taken). Sets are usually shown as circles, and any given element, x, can be seen to be a member of a set by being drawn inside it **(1)** or shown not to be a member if it is outside the circle **(2)**.

Two or more sets can be shown to intersect by shading the regions where they overlap **(3** or **5)**; shaded in their entirety, they have been unified into a single set **(4** or **6)**

HAVE YOU ORDERED YOUR VOLUME SEVEN BINDER YET?

FOLD 2

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 7.

☐ PLEASE SEND ME MY VOLUME 7 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

FOLD 3

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

NO STAMP NECESSARY.

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £_____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

FOLD 1

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

		INITIALS	SURNAME
MR			
MRS			
MISS			

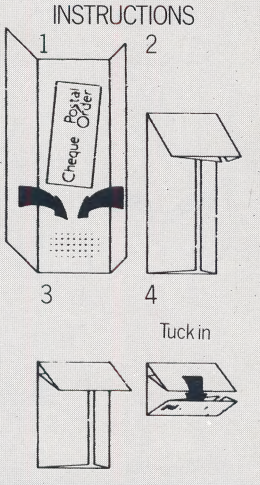
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

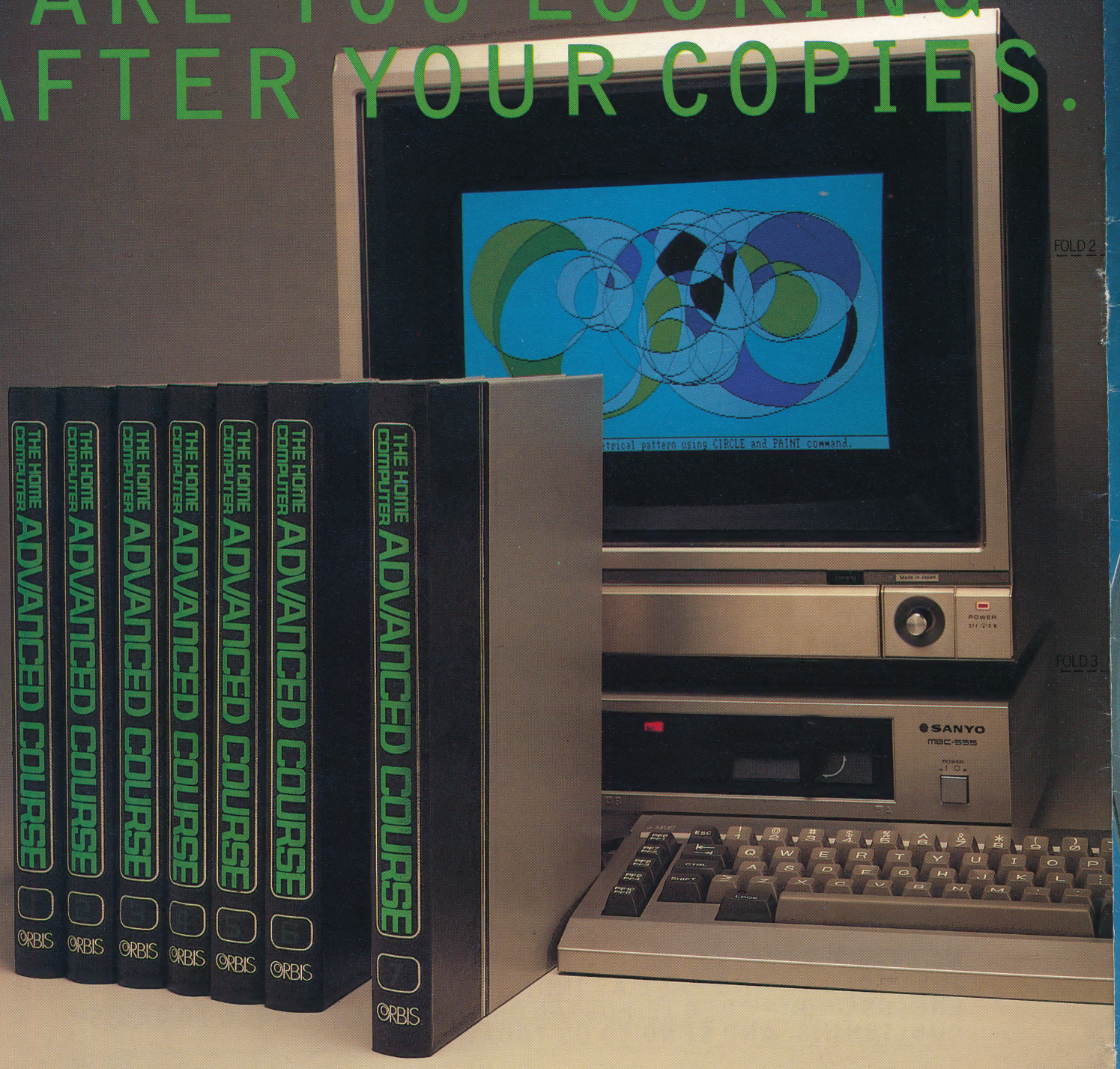
FOLD 1

FOLD 4

FOLDING INSTRUCTIONS



ARE YOU LOOKING AFTER YOUR COPIES.

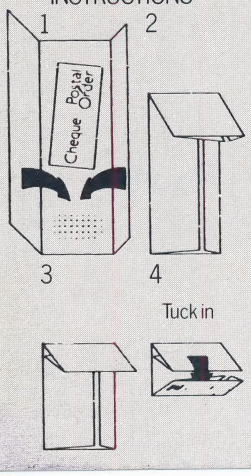


FOLD 2

FOLD 3

FOLD 4

FOLDING INSTRUCTIONS



FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

12

FOLD 1

GUARANTEE

If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.

GUARANTEE